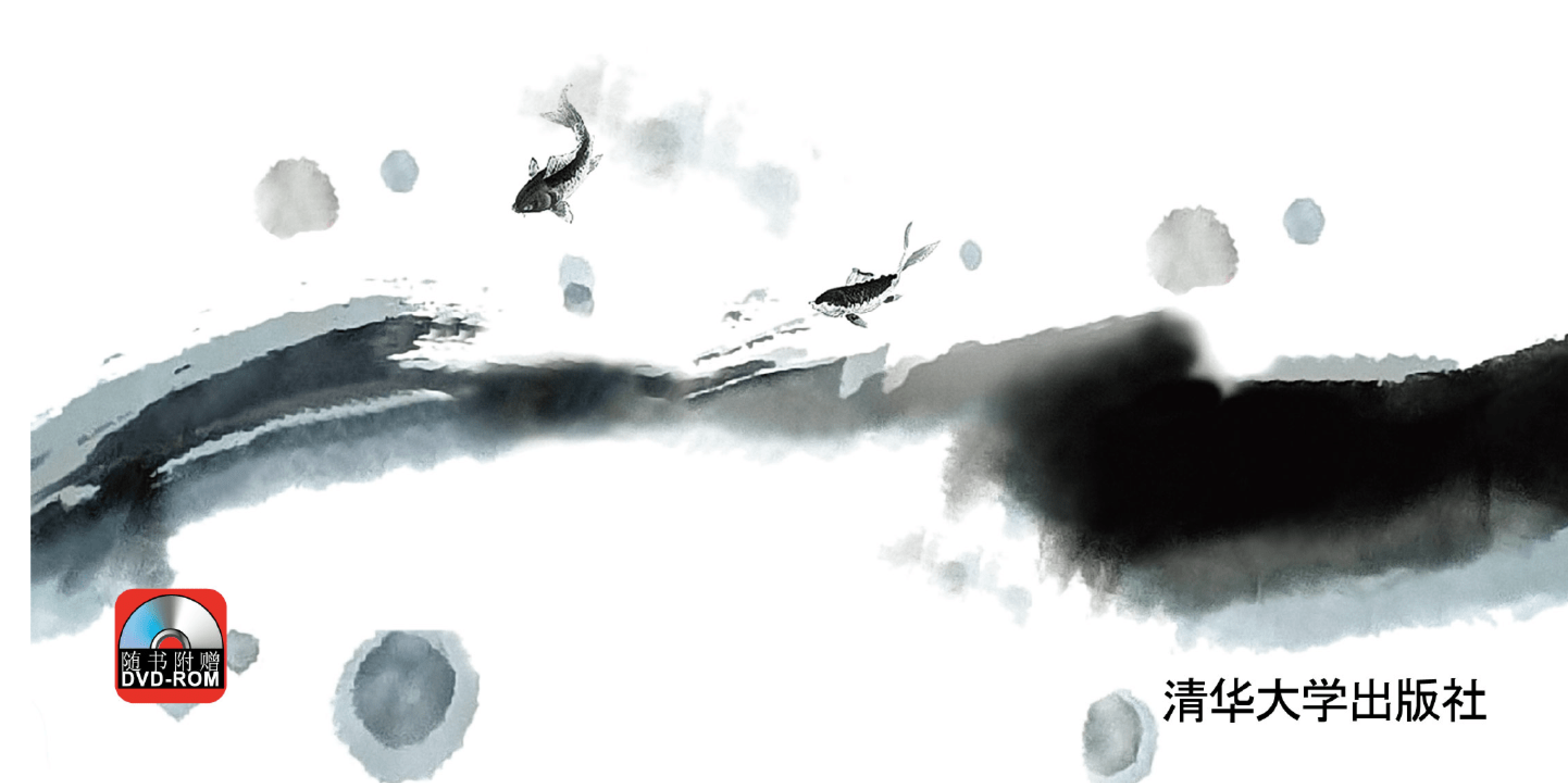
Web程序员成功之路

Web服务

开发学习实录

- 迅速提高读者Web开发能力，全面挖掘读者开发潜力
- 一线资深Web程序员经验力作，窗内网独家推荐自学教材
- 14个小时视频教学，简化学习过程
- 49个实战案例与理论知识综合讲解，提高应用能力
- 网站互动教学（www.itzcn.com），QQ群在线帮助读者解疑

闫建强　王瑞敬　编著



清华大学出版社

Web 程序员成功之路

Web 服务开发学习实录

闫建强　王瑞敬　编著

清华大学出版社

北　京

内 容 简 介

本书以 Web 服务技术的原理为主线，详细讲解在.NET 平台上的实现方式。内容涵盖 Web 服务的基础概念、核心组成部分、Web 服务的开发、Web 服务的应用、Web 服务的通信、Web 服务的安全性和集成第三方 Web 服务等。

全书概念清晰、兼顾深度和广度，采用大量的应用实例来辅助读者理解技术原理。同时读者可以通过这些实例了解 Web 服务的开发过程。

本书面向初学者，为读者了解和开发 Web 服务提供了捷径，可作为高等院校相关专业本科生和研究生的教材，也适合广大技术人员作为了解 Web 服务的参考书。

图书在版编目(CIP)数据

Web 服务开发学习实录/闫建强，王瑞敬编著. --北京：清华大学出版社，2011.8

(Web 程序员成功之路)

ISBN 978-7-302-26266-4

Ⅰ. ①W…　Ⅱ. ①闫…　②王…　Ⅲ. ①互联网络—网络服务器—程序设计　Ⅳ. ①TP368.5

中国版本图书馆 CIP 数据核字(2011)第 138539 号

责任编辑：张　瑜　杨作梅
装帧设计：杨玉兰
责任校对：李玉萍
责任印制：

出版发行：清华大学出版社　　**地　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：**100084
社 总 机：010-62770175　　**邮　购：**010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190×260　**印　张：**29.5　**字　数：**714 千字
版　　次：2011 年 8 月第 1 版　**印　次：**2011 年 8 月第 1 次印刷
印　　数：1~4000
定　　价：58.00 元

产品编号：

前　　言

随着互联网技术的不断发展，越来越多的应用程序需要集成在 Internet 中。信息在网络中进行传递，需要适应不同的网络环境，在不同的网络平台之间实现通信的兼容。

Web 服务是基于网络的一种软件开发模式。它通过规范性的设计、发布、实现和调用，可以实现多个 Web 服务构建成一个完整的应用程序功能。

本书详细介绍了 Web 服务的概念及其相关的协议和技术规范。然后通过 C#编写一些实现 Web 服务的解决方案，并使用一些针对性和应用性较强的案例来加深读者对 Web 服务的理解。

本书内容

第 1 章　Web 服务入门知识。本章主要介绍什么是 Web 服务、Web 服务的优势、Web 服务的技术架构，以及 Web 服务在.NET 平台中的简单用法。

第 2 章　构建 ASP.NET Web 服务。本章将学习创建 Web 服务的各种方法，并重点介绍用 Visual Studio 创建 ASP.NET Web 服务和修改 Web 服务属性的方法。

第 3 章　Web 服务基础知识之 XML 技术。本章详细介绍 Web 服务基础中有关 XML 的知识，包括 XML 声明、XML 属性、XML 命名空间、XML 的实体引用以及 DTD 等。

第 4 章　Web 服务类型系统——XSD。本章详细介绍在 Web 服务中检验 XML 数据正确性的一个机制 XSD，包括定义简单和复杂的数据类型、匿名类型以及验证 XSD 等。

第 5 章　Web 服务描述语言。本章将会对 WSDL 进行详细的讲解，包括如何查看 WSDL 文档、WSDL 文档的结构，以及 WSDL 的各个组成部分等。

第 6 章　简单对象访问协议——SOAP。本章深入讲解 ASP.NET Web 服务中 SOAP 协议的应用，包括 SOAP 的报头和主体、SOAP 传递消息和扩展等。

第 7 章　管理 Web 服务的状态。本章讲解在设计 Web 服务时，如何对客户端的状态信息进行保存和管理，包括使用 Session、Application 和 Cookie 等。

第 8 章　异步服务。本章将深入学习基于 ASP.NET Web 服务的同步化调用和异步化调用。

第 9 章　利用 ASP.NET 的缓存和事务功能。本章将讲解如何为 Web 服务添加缓存和事务功能，包括 ASP.NET 缓存机制、输出缓存、应用程序缓存以及事务的使用等。

第 10 章　安全性和验证。本章将深入讲解 ASP.NET 中关于 Web 服务的身份验证、授权机制，以及使用 SOAP 实现安全通信的技术，以保护 Web 服务的安全等。

第 11 章　.NET 下的 XML 操作。本章将学习如何在 ASP.NET 中读取和写入 XML、XML 的属性和节点的增删改操作，以及 XML 序列化等。

第 12 章　集成第三方 Web 服务。本章介绍如何使用网络中提供的一些现有的 Web 服务，以及把这些 Web 服务集成到一个 Web 应用程序中。

第 13 章　WCF 快速入门。本章将介绍什么是 WCF、WCF 的基本术语以及如何创建 WCF 和编写客户端代码。

第 14 章　网络聊天工具。本章我们将在 Windows Form 中使用 Web 服务来开发一个简单的网络聊天工具，实现注册、登录、添加好友、发送消息和文件以及监听等功能。

第 15 章　留言簿。本章主要介绍如何使用 ASP.NET 结合 Web 服务实现留言簿，功能包括添加留言、管理员登录、回复和删除留言等。

本书特色

本书中的大量内容来自真实的 Web 服务项目，力求通过对读者实际操作时的问题进行描述并加以解决的方法使读者更轻松地掌握 Web 服务的开发。本书难度适中，内容由浅入深、实用性强、覆盖面广、条理清晰。

1)　结构独特

通过“网络教学→基础知识→实例描述→实例应用→运行结果→实例分析”的形式将每个知识与实际应用中的问题相结合。

2)　形式新颖

用准确的语言总结概念、用直观的图示演示过程、用详细的注释解释代码、用形象的比喻帮助记忆。

3)　技术文档

技术文档包含一些非常简单的知识点或者理论性的内容。通常这些内容没有具体的应用案例，但又是读者必须要了解的，例如一些概念和术语。

4)　内容丰富

涵盖了实际开发中 Web 服务技术中涉及的 XML 技术、XSD、WSDL、SOAP、Web 服务状态、异步服务、缓存、事务、安全性和验证技术、WCF 等方面的各种问题。

5)　随书光盘

本书为实例配备了视频教学文件，读者可以通过视频文件更加直观地学习 Web 服务的使用知识。

6)　网站技术支持

读者在学习或者工作的过程中，如果遇到问题，可以直接登录 www.itzcn.com 与我们取得联系，作者会在第一时间给予帮助。

7)　贴心的提示

为了便于读者阅读，全书还穿插着一些技巧、提示等小贴士，体例约定如下。

提示：通常是一些贴心的提醒，让读者加深印象或提供建议和解决问题的方法。

注意：提出学习过程中需要特别注意的一些知识点和内容，或者相关信息。

技巧：通过简短的文字，指出知识点在应用时的一些小窍门。

读者对象

本书具有知识全面、实例精彩、指导性强的特点，力求以全面的知识性及丰富的实例来指导读者透彻地学习 Web 服务各方面的知识。本书可以作为 Web 服务的入门书籍，也可以帮助中级读者提高技能，对高级读者也有一定的启发意义。

本书适合以下人员阅读学习：

- Web 服务初学者以及在校学生
- Web 应用程序开发人员

- 各大中专院校的相关授课老师
- 其他使用 Web 服务的从业人员

参编人员

本书主要由闫建强、王瑞敬编写，其他参与编写、资料整理、程序开发的人员还有赵俊昌、张芳芳、杨飞、赵振方、胡海静、秦方、杨晓军、侯俊杰等。

由于编者水平有限，书中难免存在不足和疏漏之处，恳请读者批评指正。

目　　录



第 1 章　Web 服务入门知识

内容摘要：

互联网经过多年的发展，已经日渐普及。一提到网络，人们就会想到 Web，Web 确实为互联网的发展做出了巨大的贡献。能创造如此辉煌成就的 Web，得益于它天生的两大独特优势：简单性和方便性。作为 Web 应用的重要组成部分，Web 服务也同样继承了这两个优势。

在没有使用 Web 服务的应用程序中，不同的系统之间进行交互是非常困难的，甚至是不可能的事情。Web 服务定义了一套统一的标准，使用可扩展的标记语言 XML 进行数据通信。所以我们使用 Web 服务就可以忽略在应用程序中各系统之间的通信差异，真正实现跨平台、跨网络、跨系统、跨语言的应用程序通信功能。

Web 服务的应用范围广、使用方便，所以有人说它的出现颠覆了传统的网络应用程序通信机制，预示着一种新的应用程序架构的革命。

Web 服务是一种自包含、自描述、模块化的应用程序，它可以通过 Web 来发布、定位和调用。Web 服务的功能可以是一个很简单的请求，也可以是一个非常复杂的商业应用。不过一旦 Web 服务在 Internet 上发布以后，在 Internet 上运行的其他应用程序就可以定位并且使用它。

本章我们将介绍什么是 Web 服务、Web 服务的优势、Web 服务的技术结构，以及 Web 服务在.NET 平台中的简单用法。

学习目标：

- 了解什么是 Web 服务
- 了解 Web 服务的优势
- 了解 Web 服务的技术结构
- 简单了解.NET 平台中的 Web 服务的使用方法
- 简单了解.NET 平台中的 Web 服务的创建方法

1.1　访问网络中的 Web 服务

Web 服务是 Internet 中发布的一种数据信息交互服务，我们可以在世界上有 Internet 服务的任何地方访问 Internet 中发布的 Web 服务。

Web 服务是一种自包含、自描述、模块化的应用程序，所以可以直接访问互联网中的 Web 服务，查看它的描述和介绍等。

在本节中，我们就简单来了解一下什么是 Web 服务，以及互联网中的 Web 服务。

**视频教学：** 光盘/videos/01/访问网络中的 Web 服务.avi　　长度：13 分钟

网络发展到今天，很少有哪个互联网名词会像“Web 服务”一样这么快流行起来并引起广泛关注的。

确实，Web 服务的出现，可能会是一场软件行业的革命。不过，Web 服务到底是什么呢？

1．Web 服务的概念

Web 服务(Web Service)在本书中并不是指 Web 中的搜索引擎、网上交易、资源下载之类的服务。

本书中 Web 服务是指在网络中发布的一些应用程序接口，可以供网络中其他的应用程序访问。使用这些 Web 服务可以很轻松地构建出功能更加强大的应用程序，这些程序功能都由 Web 服务来提供。

Web 服务的出现预示着一种新的应用程序架构的出现。在可以预见的未来，互联网中的应用将以 Web 服务的形式来体现，开发应用程序的工作将转向使用 Web 服务组建应用程序来完成。

从软件开发的角度来讲，Web 服务是 Web 服务器提供的一个应用程序，或者执行代码的程序块。它通过标准的 XML 协议来展示它的功能。

2．Web 服务的作用

有些人会觉得，Web 服务与应用服务提供商(如搜索引擎、网上银行、网上购物)提供的网络服务相同。这个观点不完全对，Web 服务确实是网络中一些运营商提供的一些基于 Web 的服务。但是其只是以一种接口的形式展示的一种应用，站在程序开发人员的角度来讲，Web 服务是为程序开发人员提供的一种调用接口，并不能被最终用户直接使用。

那么，肯定还会有人提出疑问：Web 服务到底可以在什么地方使用呢？

换个角度来思考，有时候同一个功能在很多系统中都会用到，而我们不可能为每一个系统都实现这种功能。

最典型的例子如提供天气预报功能的 Web 服务。很多应用程序为了提高实用性和人性化程度，会在程序中加入天气提醒或延伸的其他功能。但是，我们不可能要求这些应用程序的所有者都自己建立一个气象台，所以我们就可以使用网络中专门进行气象预测功能的运营商提供的 Web 服务来实现相应的功能，如图 1-1 所示。

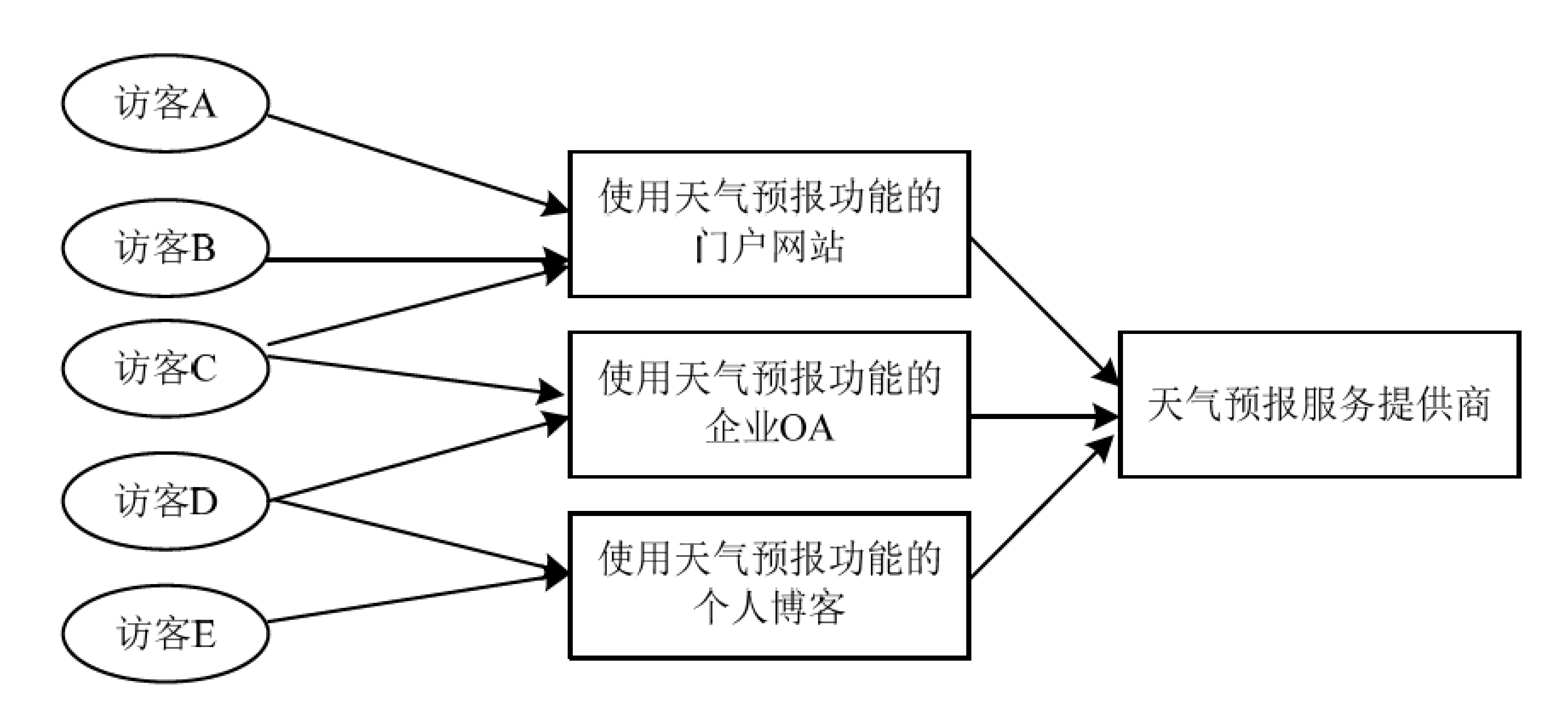


图 1-1　Web 服务示意图



当然，在网络中提供这些服务的提供商不是慈善机构，可能会收取一定的费用来为 Web 服务者提供相应的支持。

换句话来说，Web 服务只是一种共享编程的方法，我们可以把它看作“应用于 Web 中的 COM(Component Object Model，组件对象模型)”，只不过它们的实现技术有些差别。

当然，我们除了使用互联网中 Web 服务提供商提供的 Web 服务外，还可以自己开发并发布一些 Web 服务供互联网中的用户使用，也可以作为一种低耦合的程序设计方案来进行处理。

下面列出一些常用的 Web 服务的使用场合。

- 企业对企业之间的内部数据交互系统。这可能是当前使用 Web 服务最常见的场合。Web 服务允许文档和知识共享，或者相关的服务的集成。比如 Web 服务可以帮助电子商务公司与货运公司的系统相关联，实现自动填写货运申请提交到货运公司的系统中。
- 作为开发人员的预创建模块。比如，第三方的 Web 服务提供商可以创建用于认证的 Web 服务，供其他应用程序使用。
- 作为分布式应用程序的交互接口。比如我们开发了一个分布式的网络应用系统，在系统的各个部分与服务器之间进行数据交互，选择 Web 服务的方式将是非常简便的一种方式。
- 作为跨平台应用程序的核心组件。比如我们开发了一个可以运行于包括台式机、掌上电脑、手机等上的网络程序，使用 Web 服务将会是一种不错的解决方案。你不必担心不同平台中程序的数据包兼容性问题，而只确保每一种用户都能连入 Internet 就可以了。
- 作为同一家企业中的不同系统之间的连接工具来使用。比如我们可以使用 Web 服务将一个企业销售管理系统和人力资源管理系统(HR)相关联。

或许通过这些讲解你已经对 Web 服务有一些认识了，或许你也有把 Web 服务组织到自己现有的系统中的想法了。

不过要深刻地理解 Web 服务，还需要牢记这个非常重要的概念：Web 服务不是最终的用户产品，而是类似于组件的应用程序，Web 服务允许在不同环境和不同的客户端重用业务逻辑。

Web 服务的使用者永远是另一个应用程序。

3．实例描述

Web 服务是网络中的一些公开程序接口，也就是说可以在 Internet 的任意地方访问它们。本实例，我们将在网络中查找一个 Web 服务，来简单了解这个 Web 服务的用法。

4．实例应用

【例 1-1】 访问网络中的 Web 服务

(1) 首先，要到 Internet 上寻找一个提供 Web 服务的网站。这里我们使用一家国内 Web 服务网站(网址：http://www.webxml.com.cn)提供的一些免费的 Web 服务，如图 1-2 所示。



图 1-2　Web 服务页面

(2) 页面中有一个中英文双向翻译功能的 Web 服务(网址：http://fy.webxml.com.cn/webservices/EnglishChinese.asmx)，我们可以用它来实现中英文翻译的功能。访问该 Web 服务，结果如图 1-3 所示。



图 1-3　中英文翻译 Web 服务

从图 1-3 中可以看到，该页面以列表的形式展示了该 Web 服务下所有可用的操作。

(3) 接下来单击执行中英文双向翻译功能的 Translator 链接，打开功能的调用页面，如图 1-4 所示。

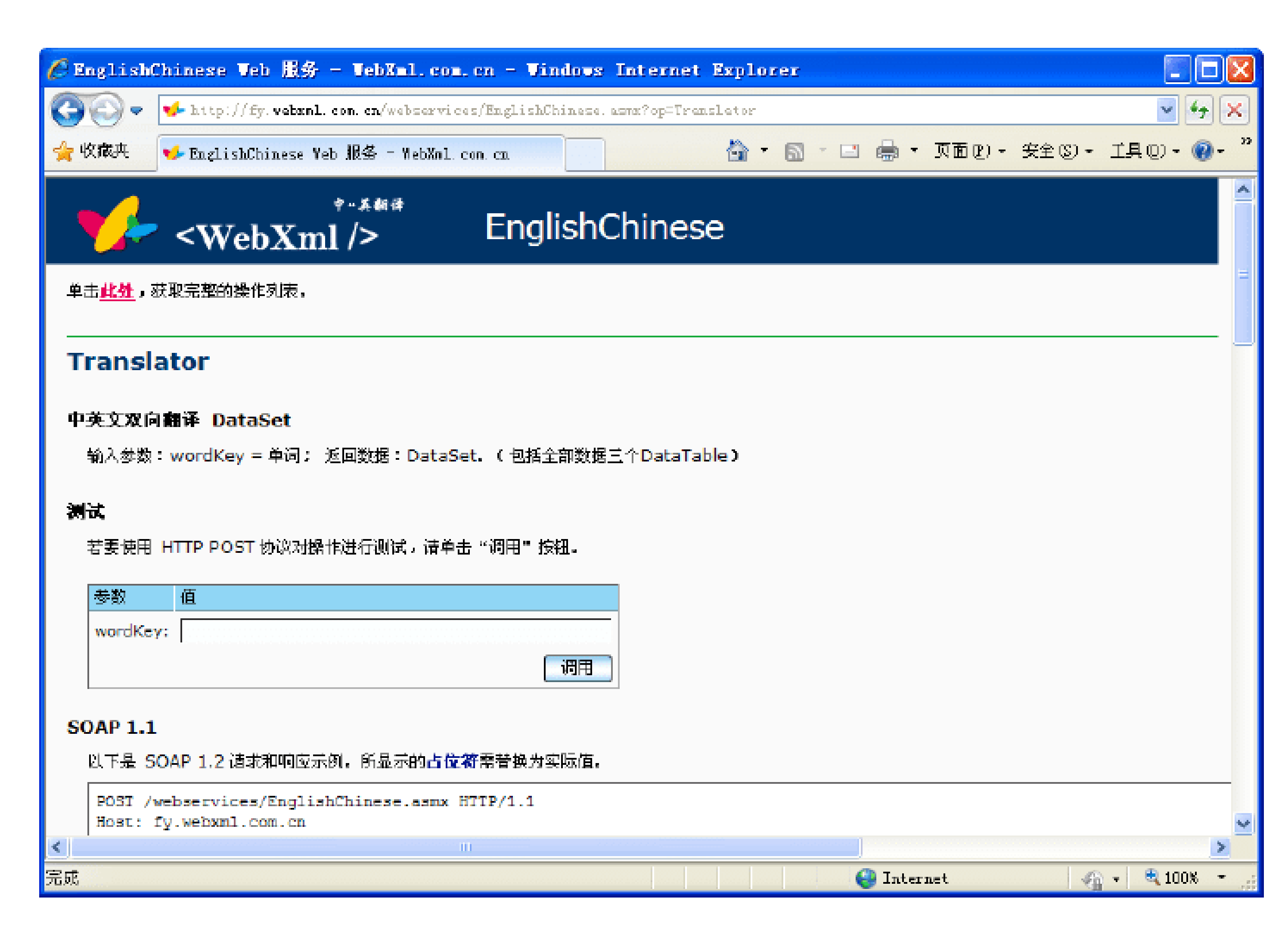


图 1-4 中英文双向翻译

从图 1-4 中可以看到，系统列出了该操作的简单说明、测试接口、一些请求和响应的示例(因为浏览器大小的原因，截图中没有全部显示)。

(4) 可以在 wordKey 文本框中输入要翻译的单词“Hello”，并单击【调用】按钮，系统将返回一段包含响应结果的 XML 文本，如图 1-5 所示。

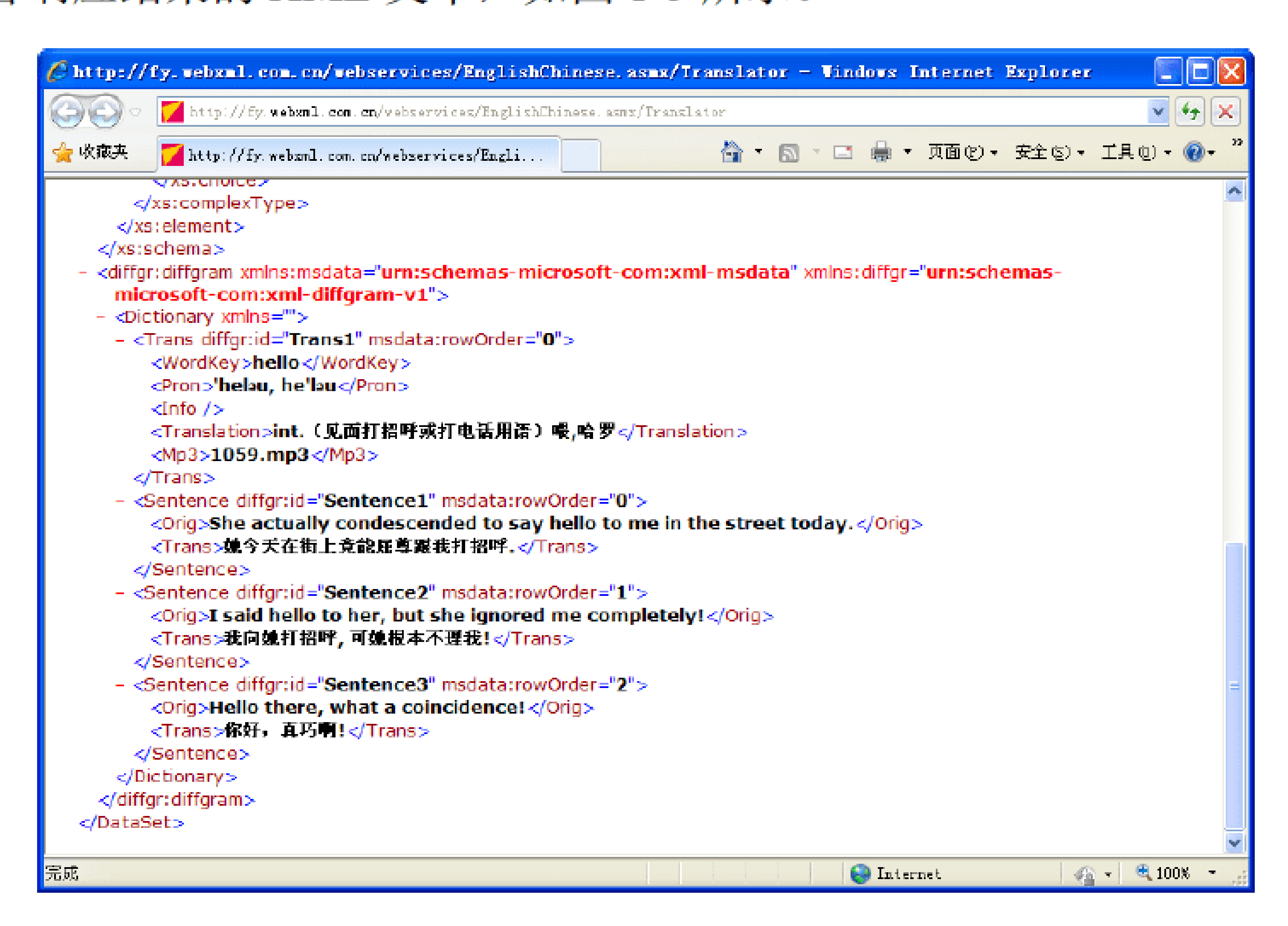


图 1-5 翻译请求响应结果

从图 1-5 中可以看到，这段 XML 文本中包含了我们想要的翻译结果，包括该单词的音标、中文意思以及一些简单的应用示例。

因为任何编程语言都可以处理 XML 格式的文本，所以只要能够获取这段 XML 文本，我们就可以在任何一种语言或平台中解析并使用它。

5．实例分析



源码解析：

本实例，我们首先在 Internet 中搜索找到一个提供在线翻译功能的 Web 服务的 Web 服务器。然后找到在线翻译功能的 Web 服务链接地址，并访问该地址，选择执行在线翻译功能的 Translator 操作，打开 Web 服务测试页面，在该页面我们可以看到该 Web 服务操作的使用说明。我们只需要在 wordKey 文本框中输入要查询的单词并单击【调用】按钮就可以获得以 XML 格式显示的翻译结果。

1.2　在.NET 中使用 Web 服务

从上一节的实例中，我们简单地了解了 Web 服务。不过既然前面着重强调“Web 服务的使用者永远是另一个应用程序”这一根本思想，所以作为一名开发人员，你一定会追切地想知道在 Web 应用程序中如何使用这些 Web 服务。

本节，我们就在.NET 语言中调用网络中提供的一个 Web 服务，来简单了解.NET 语言中的 Web 服务的用法。



视频教学：光盘/videos/01/在.NET 里使用 Web 服务.avi　　长度：9 分钟

1.2.1　实例描述

在前面提到的提供 Web 服务的网站 http://www.webxml.com.cn 中的 Web 服务列表页面还有一个查询 QQ 在线状态的 Web 服务，它也是一个免费的功能。

本实例，我们就以查询 QQ 在线状态为例演示.NET 中的 Web 服务的用法。

1.2.2　实例应用

【例 1-2】 在.NET 里使用 Web 服务

(1) 首先，找到一个提供 QQ 在线状态查询的 Web 服务，网址为 http://webservice.webxml.com.cn/webservices/qqOnlineWebService.asmx。

(2) 在浏览器中输入这个网址来测试 Web 服务，如图 1-6 所示。

从图 1-6 所示的页面可以知道，该 Web 服务只提供了一个名为 qqCheckOnline 的操作，并且该操作接收一个 QQ 号码作为参数并返回相应的状态码(或错误码)。

(3) 单击 qqCheckOnline 链接先来测试一下这个操作，如图 1-7 所示。



图 1-6　QQ 在线状态查询的 Web 服务



图 1-7　qqCheckOnline 操作测试

(4)　在 qqCode 文本框中输入一个 QQ 号码，然后单击【调用】按钮，系统将返回请求到的状态信息，如图 1-8 所示。

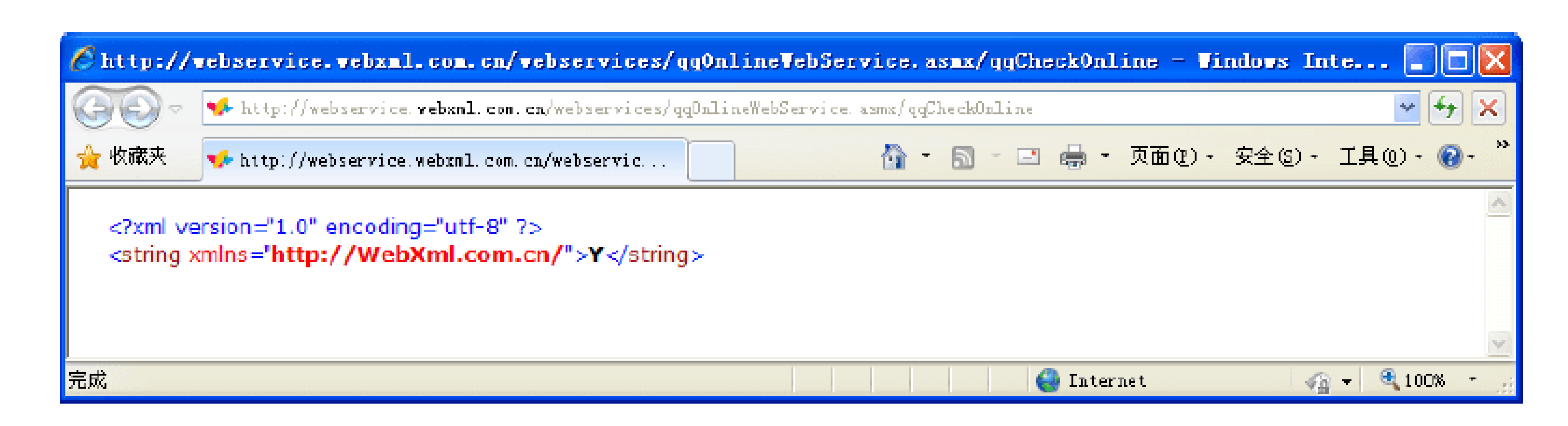


图 1-8　获取 QQ 在线状态

至此，说明这个查询 QQ 在线状态的 Web 服务是可用的，下面我们使用一个控制台应用程序演示.NET 中该 Web 服务的用法。

(5)　打开 Visual Studio 2010 创建一个控制台应用程序，命名为 QQOnline。

(6)　鼠标右键单击项目名称 QQOnline，在弹出的快捷菜单中选择【添加服务引用】命令，打开【添加服务引用】对话框，如图 1-9 所示。

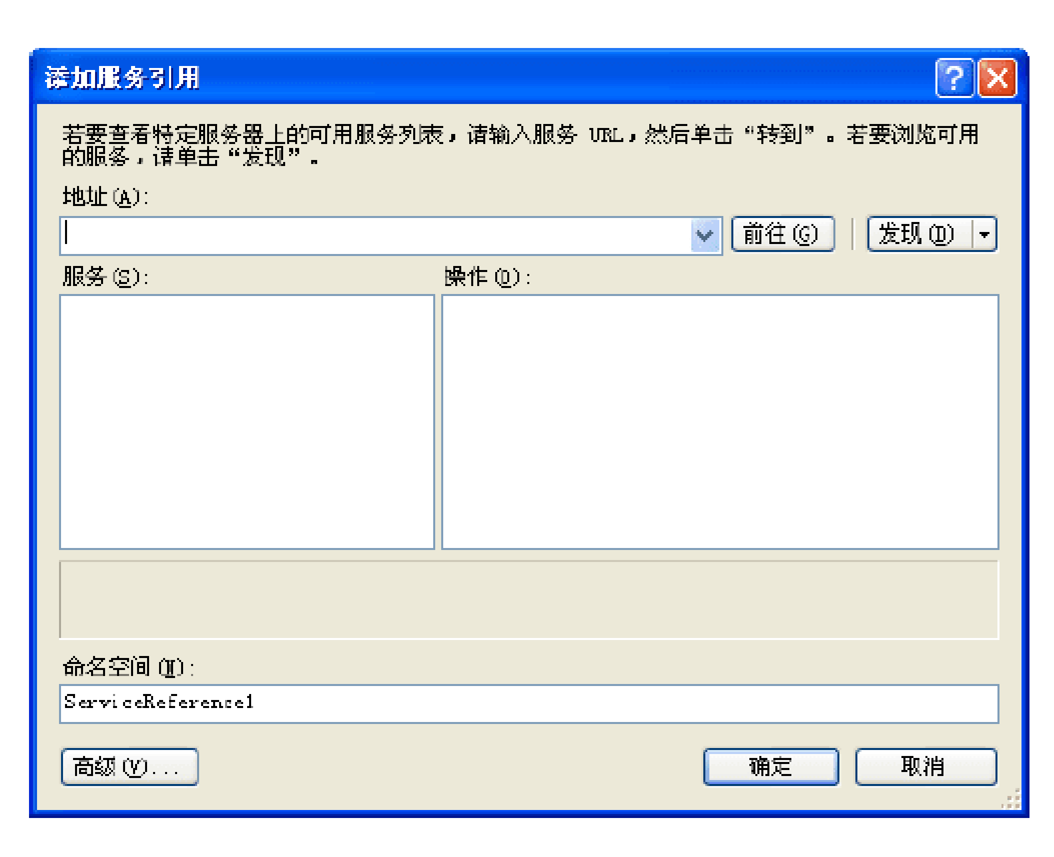


图 1-9 【添加服务引用】对话框

(7) 在【添加服务引用】对话框的【地址】下拉列表框中输入 Web 服务的地址“http://webservice.webxml.com.cn/webservices/qqOnlineWebService.asmx”，然后单击【前往】按钮。稍等片刻就会链接成功，在【服务】列表框中就会列出相应的请求结果，如图 1-10 所示。



在单击【前往】按钮以后，如果网速正常，几秒或几十秒就可以获得响应，如果网速较慢，可能会需要较长的时间。

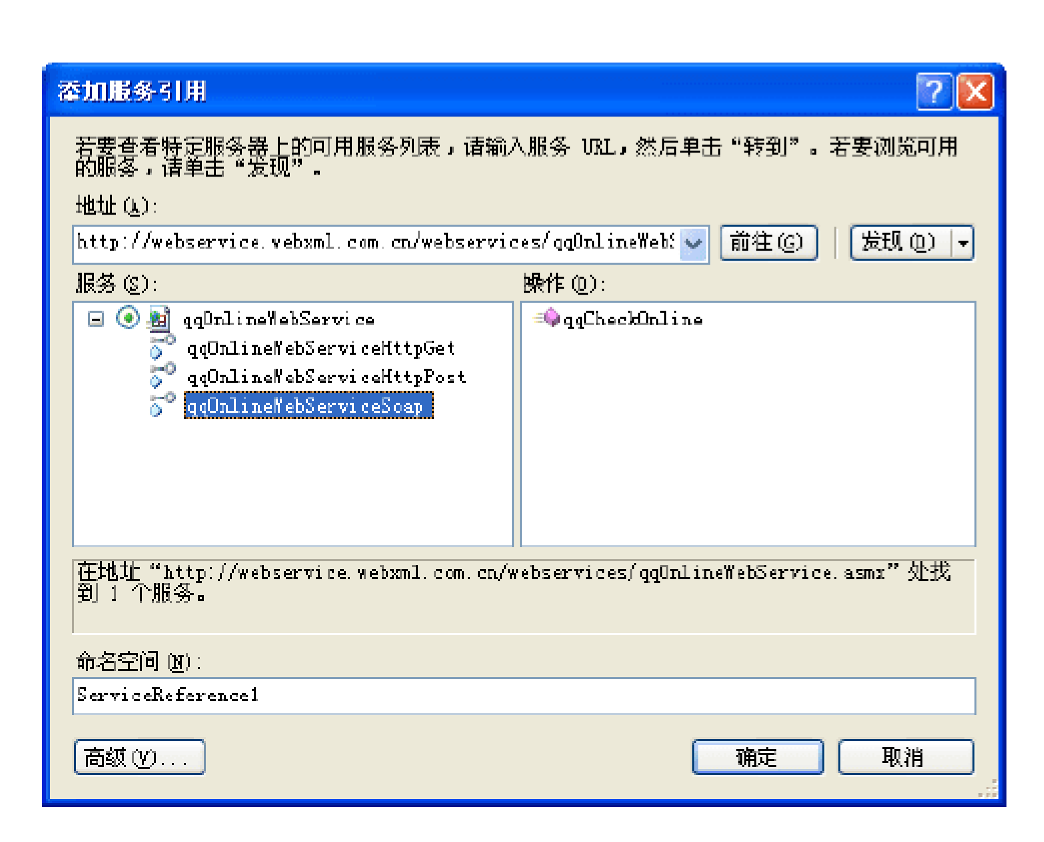


图 1-10 添加服务引用

(8) 修改【命名空间】文本框中的值为 QQOnlineQuery，然后单击【确定】按钮。这样就成功地添加了查询 QQ 是否在线 Web 服务的引用。



在使用 Visual Studio 2010 添加对 Internet 中 Web 服务的引用时，系统可能会在应用程序配置文件(web.config 或 app.config)中生成一些重复的配置节点，这时就需要我们手动删除多余的配置节点。

(9) 接下来修改控制台 Program 类中的代码，如下所示：

```
class Program
{
    static void Main(string[] args)
    {
```

```
            QQOnlineQuery.qqOnlineWebServiceSoapClient onlineQuery =
                new QQOnlineQuery.qqOnlineWebServiceSoapClient();

            Console.Write("请输入您要查询的 QQ 号码：");
            string inputQQ = Console.ReadLine();
            string status = onlineQuery.qqCheckOnline(inputQQ);
            PrintStatus(status);
            Console.ReadLine(); //程序暂停，等待用户回车确认
        }
        static void PrintStatus(string status)
        {
            switch (status)
            {
                case "Y":
                    Console.WriteLine("在线");
                    break;
                case "N":
                    Console.WriteLine("离线");
                    break;
                case "E":
                    Console.WriteLine("QQ 号码错误");
                    break;
                case "A":
                    Console.WriteLine("商业用户验证失败");
                    break;
                case "V":
                    Console.WriteLine("免费用户超过数量");
                    break;
            }
        }
    }
```

这里我们创建一个查询 QQ 号码在线状态的实例对象 onlineQuery，然后调用该对象的 qqCheckOnline()方法查询用户输入的 QQ 号码在线状态，并将请求结果打印到控制台中。

1.2.3　运行结果

保存程序文件。单击工具栏上的【启动调试】按钮，程序会提示输入一个 QQ 号码，执行结果如图 1-11 所示。



图 1-11　查询 QQ 号码的在线状态

1.2.4　实例分析



源码解析：

本实例，我们首先获取查询 QQ 在线状态的 Web 服务地址，然后在浏览器中访问该地址，使用一个 QQ 号码进行测试。我们需要在控制台程序中添加对该 Web 服务的引用。当成功添加引用以后，就可以使用引用时输入的命名空间下的 qqOnlineWebService SoapClient 类创建查询对象。之后，通过调用该对象下的 qqCheckOnline()方法完成相应的查询功能。

1.3　为什么使用 Web 服务

前面列举了 Web 服务的作用，并且通过一些例子进行了简单的应用。从这些例子中我们不难看出，Web 服务在各个方面都非常有用。

从使用范围上来讲，Web 服务既可以作为一些应用服务发布给开发人员；也可以作为信息发布的接口，供用户调用。

而且，从使用者的投资成本和回报率上来讲，Web 服务十分节约成本，使用起来也非常安全方便。

下面我们就针对 Web 服务的各种优点来进行详细介绍。



视频教学：光盘/videos/01/为什么使用 Web 服务.avi　　长度：9 分钟

1.3.1　基础知识——Web 服务的优点

整体来说，Web 服务的优点非常多，以至于有人认为“它的出现颠覆了传统的网络应用程序通信机制”。

1．Web 服务是一种优秀的分布式计算技术

基于组件的分布式计算的应用程序系统通常使用的协议是 CORBA(Common Object Request Broker Architecture，通用对象请求代理体系结构)和 Microsoft 的 DCOM(Distributed Component Object Model，分布式组件对象模型)。

对于这两种协议，尽管它们有许多相通之处，但是它们在细节上有很多不同，也导致使用这两个协议的应用程序很难进行相互通信。假如要实现一个跨平台、跨 Internet、适应 Internet 的可伸缩性的应用程序时使用这两种协议，将会显得力不从心。

而 Web 服务不同，它是基于组件的分布式技术变革的必然产物。创建跨平台、跨 Internet 的分布式系统，正是 Web 服务的优势所在。

总体来说，Web 服务在实现分布式应用方面，主要有以下几个特征。

- Web 服务与客户端的联系松散。客户端向 Web 服务发出请求，Web 服务器向客户端返回响应结果，然后连接就会断开。使用这种方式不存在永久性链接，因而避免了链接管理等复杂性的问题。Web 服务可以随意扩展其接口，并在添加新的方法以后不会影响客户端的使用。
- Web 服务与状态无关。Web 服务不支持代表客户端的状态信息，这使得它在面对许多客户端的时候伸缩自如。Web 服务使用的 HTTP 协议也是与状态无关的。

2．可以轻松地跨过防火墙

对于在网络中运行的各个主机来说，引入防火墙是非常有必要的事情，防火墙可以阻止入侵者进入网络，消除局域网中不友好的网络通信。特别是对于服务器来说，使用防火墙是很有必要的事情。

但是，每一种事物的出现都有利有弊。防火墙可以很好地保护网络中主机的安全，但是这种安全保护却要付出一些代价：它使得合理的网络通信也出现一定的瓶颈。合理、正确地配置防火墙来支持 DCOM 通信是一件非常不容易的事情，所以使用 DCOM 等协议来实现一个网络通信系统非常不方便。

而 SOAP 协议从设计的角度就直接避免了这个问题，它直接标准化了对象请求时的 HTTP 使用方法。



SOAP 使用 HTTP 作为其通信协议，HTTP 协议可以很轻松地通过防火墙，所以 SOAP 就轻松地避免了这个问题。

3．使用的 SOAP 协议非常简单

虽然 DCOM 使用起来也具有一定的优势，比如其具有位置透明性等，程序中不需要包括任何特殊的 DCOM 代码等。但是建立一个 DCOM 系统并使之正常运行并不是一件容易的事情(如何通过防火墙就是一个问题)。

DCOM 服务器必须在客户端上进行注册，当服务器被修改以后，客户端的代码必须随之更新，所以基于其开发的应用程序使用起来非常不易。

使用 Web 服务就大不一样了。Web 服务使用的是 SOAP 协议，它比较灵活，要求的技术含量也非常低。SOAP 可以利用通用的 XML 分析器和当前流行的 HTTP 服务器来实现。

而且 Web 服务的串行化格式基于 XML 语言。XML 简单、可扩展、易读，已经得到广泛的支持和应用。相对于具有复杂格式的 DCOM 和 CORBA 来说，Web 服务在对传输的数据进行处理的问题上非常优秀。

4．集中信息

有人说 Web 服务的出现是“一种新的应用程序架构的革命”，很大程度上也体现在这个方面。

Web 可能真的会改变当前的应用程序的架构。以前面举过的天气预报的例子为样本，我们直接使用 Web 服务提供商提供的天气预报服务，别人也可以直接使用这种服务。那么，当有一天，所有的人都在使用这一家 Web 服务提供商的天气预报功能的 Web 服务时，全世界的天气预报信息都将集中在这一家信息中心的服务器中，就达到了天气预报信息的集中化。

当然，互联网中的其他服务也可以实现真正的集中，比如个人信息。现在的情况是每一个互联网系统都有一套用户信息系统，这样用户信息非常分散，用户在某一个系统中注册的用户信息不能在另一个系统中使用，使用多个系统的用户通常在修改一项个人信息时，需要重复修改多个系统中的相应信息。

再考虑另一种情况，很多时候我们会将自己的通信记录保存在不同的地方：有的保存在笔记本电脑中，有的保存在手机中。这些不同地方保存的信息往往不会同步，而且格式可能也不相同。

而使用 Web 服务，数据的集中处理将变得非常容易。我们可以在一个集中的信息登记处来注册个人信息，然后便可以在多个系统(地方)同步使用、更新和维护一份相同的信息。



当然这样会面临数据覆盖的风险，但是我们可以使用历史记录等工具来记录每一次修改，以备不测。

这个思想，虽然非常有创意，但是现实的情况并不能很好地适应当前的社会环境。不过庆幸的是有一些互联网服务提供商已经开始着手做此类的一些服务了，最典型的当属 Microsoft 的 Passport 了。

Passport 服务是一种单项签名验证的服务，当前它已经有很多用户了，当然主要应用在流行的 Hotmail 和 MSN 服务等系统中。

当然，随着时代的发展，这种使用 Web 服务的数据集中的形式肯定会慢慢地被用户所接受。这些互联网服务提供商的大力推广和大胆应用肯定会促进这一过程的发展。期待那一刻！

1.3.2 实例描述

刚才我们简单介绍了在实现应用程序功能的时候使用 Web 服务的诸多优点，或许我们对这些优点还没有一个很完整的概念。

下面以使用一个火车时刻表的 Web 服务为例，简单介绍一下使用 Web 服务的优点。

1.3.3 实例应用

【例 1-3】 为什么使用 Web 服务

(1) 首先，我们需要找到提供火车时刻表查询功能 Web 服务的地址，这里以 http://webservice.webxml.com.cn/WebServices/TrainTimeWebService.asmx 为例。

其实，就从这个简单的 Web 服务地址，我们就可以知道：Web 服务是一个兼容性很好的网络通信方式。

该 Web 服务地址其实就是一个 Web 服务器上的资源地址。当然，这里使用的是 Web 服务器处理的请求，使用的程序端口为 Web 服务器的端口，协议为 Web 服务器所使用的 HTTP 协议。

Web 服务对于网络中的服务器来说，应该是必开的一个服务了吧，不论是 Windows 平台还是 Linux 平台，或者是其他平台。所以，它也就没有了使用平台的兼容性的问题了。

(2) 我们使用浏览器访问这个 Web 服务的地址，结果如图 1-12 所示。



图 1-12　火车时刻表查询

(3) 我们选择通过发车站和到达站查询火车时刻表的链接 getStationAndTimeByStationName，打开该方法的测试和说明页面。

该页面中有此方法的使用说明和操作接口。使用 SOAP 请求和响应的数据格式说明如图 1-13 所示。

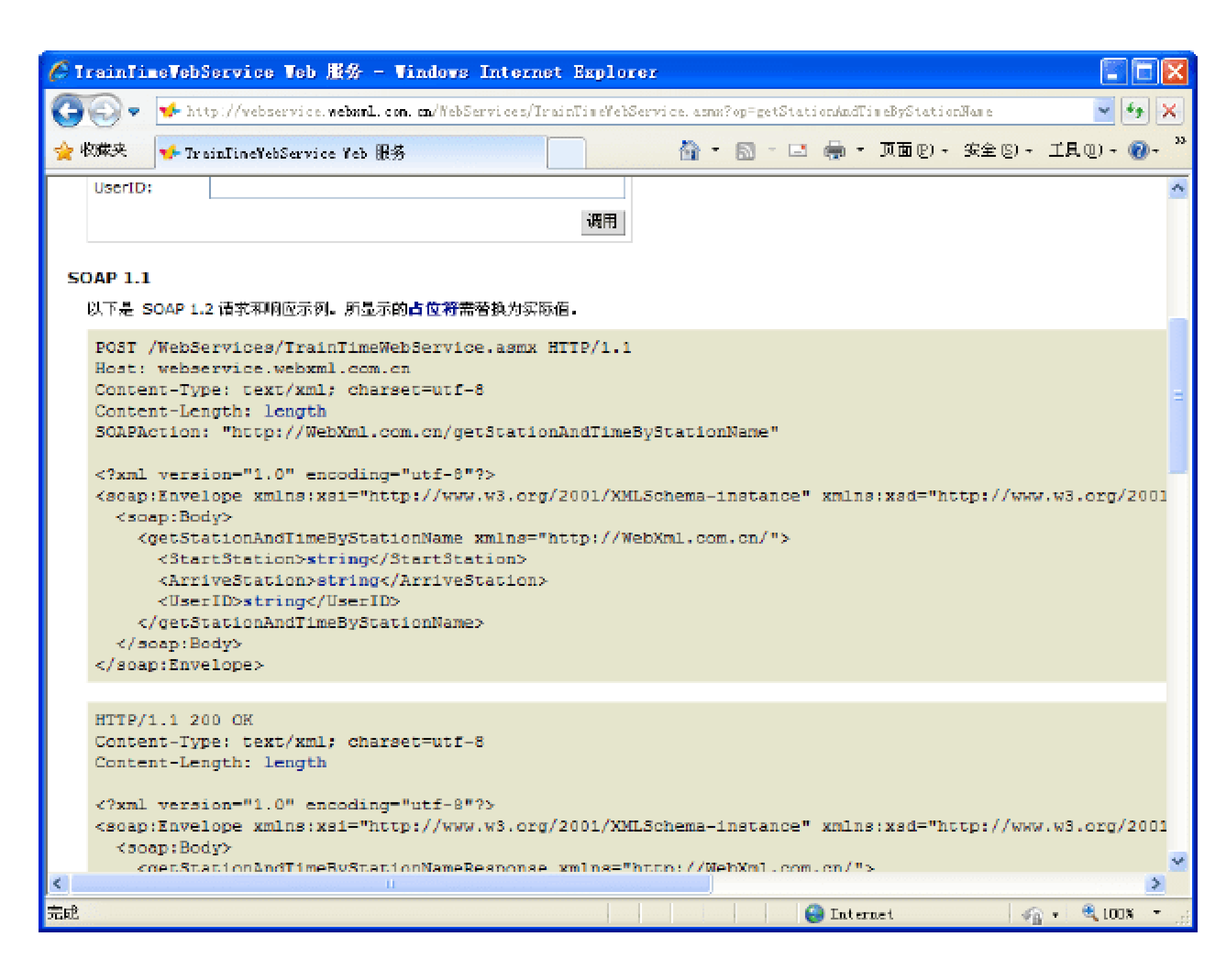


图 1-13　SOAP 请求和响应的数据格式

从图 1-13 中的 SOAP 协议请求和响应的数据信息来看，Web 服务使用 XML 来格式化数据。因为 XML 语言使用统一的标准，而且使用起来非常简单，所以基于 SOAP 的 Web 服务也非常简单易用。

(4) 我们使用该页面提供的查询接口进行简单的测试。在 StartStation 文本框中输入发车站“郑州”，在 ArriveStation 文本框中输出到达站“深圳”，然后单击【调用】按钮，将自动在新窗口打开返回的查询结果，如图 1-14 所示。

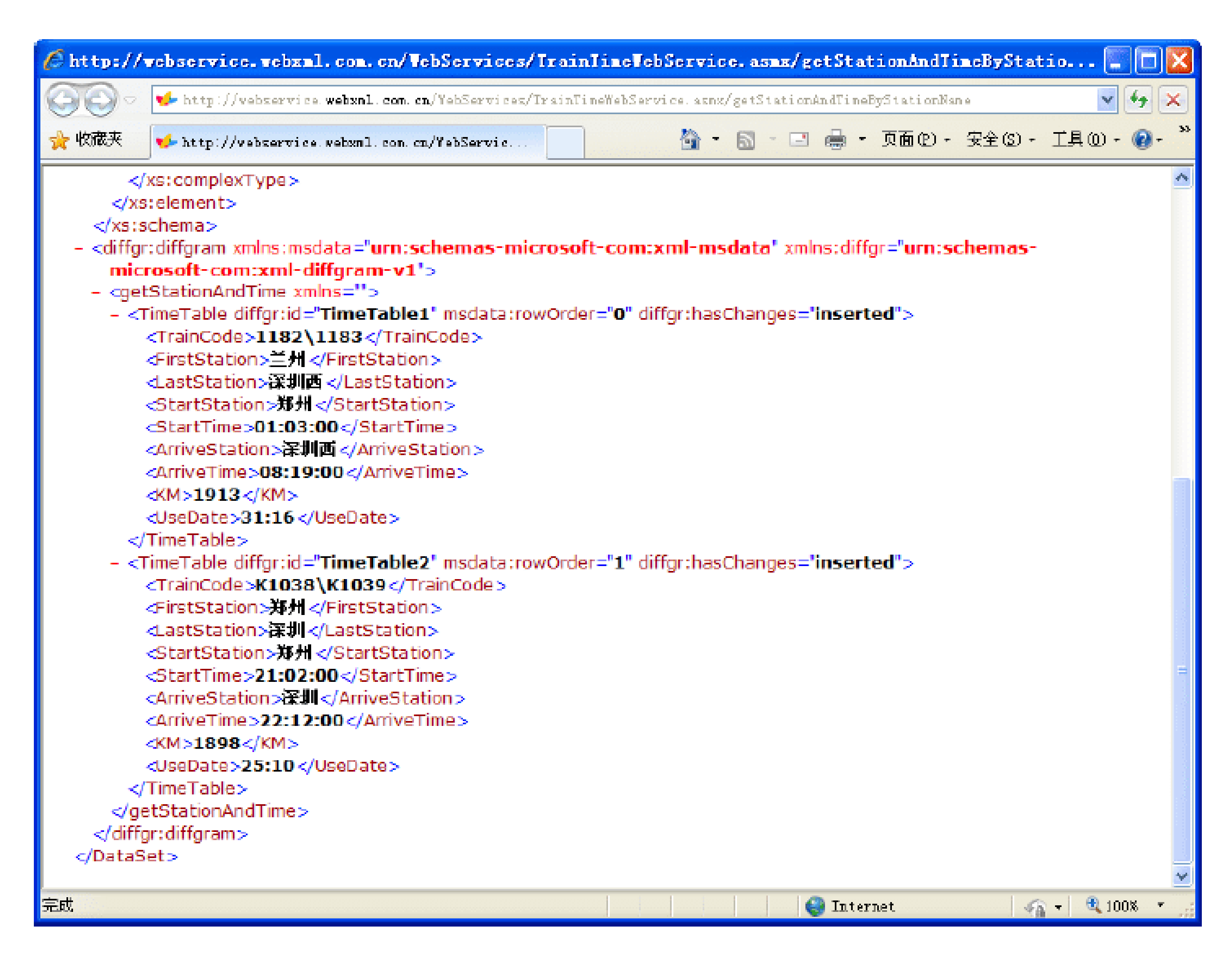


图 1-14 查询结果

从图 1-14 中可以看到，返回结果是以 XML 格式保存的。因此，在任何一个平台上，在任何一种编程语言中，我们都可以很轻松地对请求得到的 XML 文件进行解析。

另外，使用 Web 服务对火车时刻表查询的功能进行了封装、发布，使我们在 Internet 的任一地方都可以很方便地使用它，轻松地实现了数据信息服务的集中。

1.3.4 实例分析



源码解析：

本实例，我们首先获取查询火车时刻表信息的 Web 服务地址，从服务地址中可以知道 Web 服务是基于使用 HTTP 协议的 Web 应用。这样就轻松地避免了服务器安全性管理的问题。接着在浏览器中访问该地址，从页面中对 Web 服务方法的说明可以知道 Web 服务使用 SOAP 协议进行数据封装和传输，这样又解决了数据信息的跨平台性和通用性。从测试接口中，我们很方便地查询到了想要的数据，并不需要我们进行复杂的操作，这也正体现了 Web 服务的信息集中的特点。

1.4 Web 服务的技术架构

前面我们简单地对 Web 服务进行了介绍，或许我们已经对它有一定的认识了。但是，对它使用的技术、它的运行机制等问题，在我们脑海中并没有一个完整的概念。

本节，我们就来对 Web 服务所涵盖的技术进行逐一分析，对 Web 服务进行一个更加全面的了解。

视频教学：光盘/videos/01/进一步了解 Web 服务.avi　　长度：8 分钟

其实对于 Web 服务来说，它虽然是一种使用 Web 提供服务的技术，但其并不是独立的一种技术，而是多种不同技术的综合运用。

前面我们已经简单地使用.NET 访问过 Web 服务。在.NET 中，使用 Web 服务就像访问一个本地的类一样方便。那么，在这整个过程中 Web 服务是如何封装资源、如何描述、如何发现、如何访问资源的呢？

在不同类型的系统、不同的平台间实现互操作性，必须要有一套信息传输标准。对于 Web 服务来说，同样也有一套这样的标准。这套标准中包含这样一些技术：XML、SOAP、WSDL、UDDI、远程过程调用(RPC)与消息传递等。Web 服务应用程序使用这些技术执行客户端与服务器端的交互。具体执行过程如图 1-15 所示。

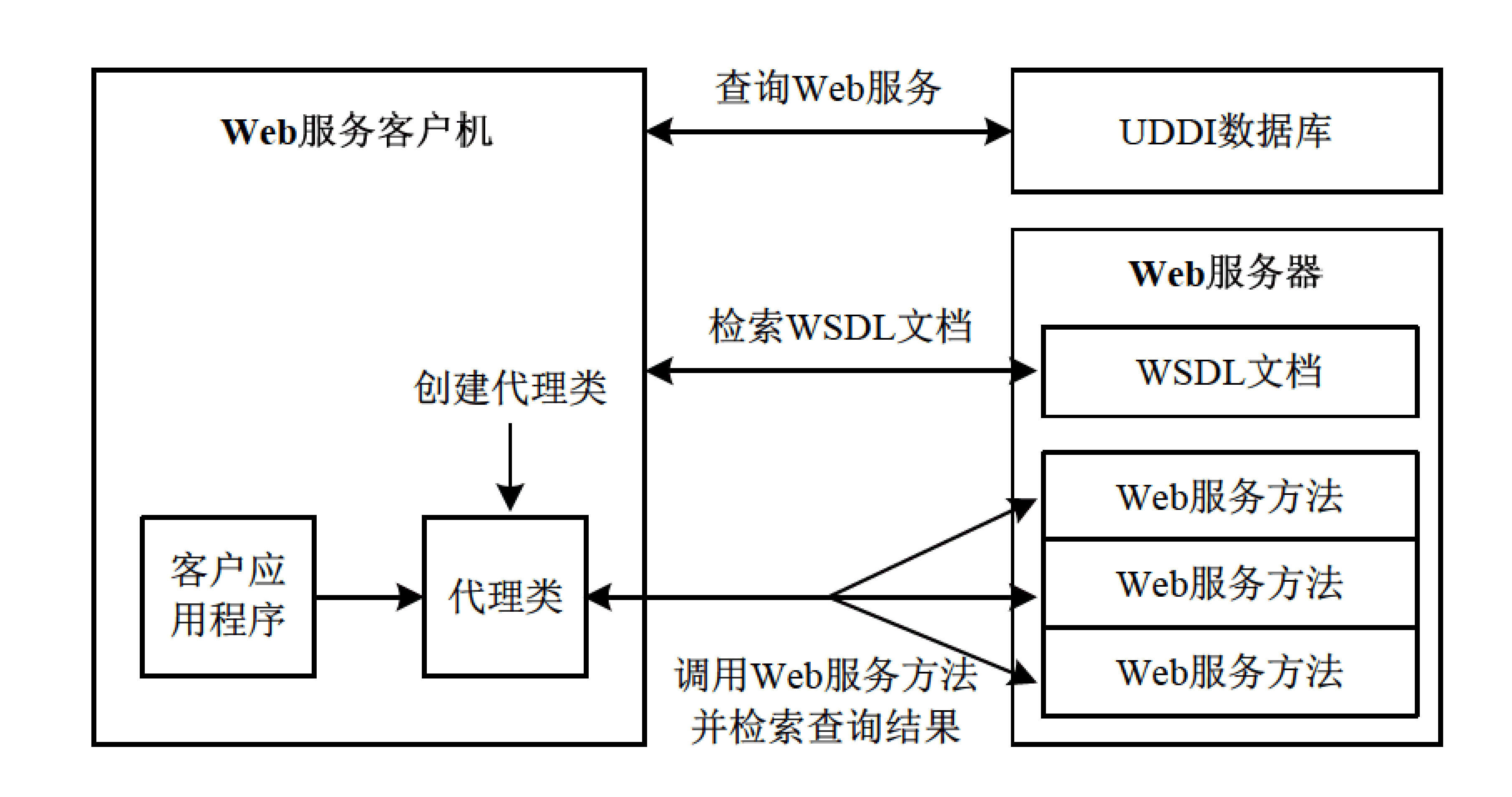


图 1-15　Web 服务的交互过程

1．XML 和 XSD

XML(Extensible Markup Language，可扩展标记语言)是万维网协会(World Wide Web Consortium，W3C)于 1998 年推荐使用的语言。经过这么多年的发展，XML 已经被广泛地接受为用于不同计算机系统的互操作性解决方案。

尽管 XML 也存在一些不足，但是由于它良好的性能已经被广为接受，所以它现在是我们追求的处理软件互操作性的最佳解决方案。

XML 是 Web 服务中表示和封装数据的基本格式。除了其容易建立和容易分析之外，XML 主要还有与平台无关、与厂商无关的特点。

Web 服务的平台是使用 XSD 来作为数据类型系统的。当用某一种语言构造一个 Web 服务时，为了符合 Web 服务的标准，所有使用的数据类型必须被转换为一个 XSD 类型。

另外，如果转换为 XSD 类型的数据需要在不同的平台和不同软件之间传递时，还需要使用某种东西将它们封装起来。这种封装它们的东西就是 SOAP 协议。

2．SOAP

SOAP(Simple Object Access Protocol，简单对象访问协议)是为 Web 服务所选择的消息传递协议。

Web 服务使用 XML 格式的消息与客户通信，而 SOAP 协议标准的核心思想是应该使用一种标准化的 XML 格式对消息进行编码。

SOAP 可以运行在任何传输协议上，如超文本传输协议(Hyper Text Transfer Protocol，HTTP)、简单邮件传输协议(Simple Mail Transfer Protocol，SMTP)等。

Web 服务希望实现不同的系统之间能够用软件对软件的方式进行对话，其打破了传统的软件应用、网站和各种不同设备间互不相容的状态，实现了基于 Web 的“无缝集成”的目的。

下面的代码展示了一个 IP 地址查询的 Web 服务的 SOAP 请求数据包的格式。该数据包中的有效数据部分包括在名为 soap:Envelope 的 XML 节点中，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetGeoIP xmlns="http://www.webservicex.net/">
      <IPAddress>string</IPAddress>
    </GetGeoIP>
  </soap:Body>
</soap:Envelope>
```

上面这段代码足以证明 SOAP 的简洁性和通用性。所以我们没有理由在 Web 服务中不使用 SOAP 数据格式。

SOAP 的优点很多，其中包括以下最为重要的三点。

- 跨平台支持：SOAP 是一段文本代码，所以可以很轻松地在各种不同的平台下使用它，当然也可以在任何一种传输协议中发送 SOAP 内容。
- 支持标题和扩展名：使用 SOAP 数据时，还可以使用工具很轻松地添加追踪、加密和安全性等特性。
- 灵活的数据类型：SOAP 允许在 XML 中对数据结构和 DataSet 编码，就像是编程语言中的简单数据类型(如数值型和字符型)一样。

3．WSDL

WSDL(Web Services Description Language，Web 服务描述语言)是使用计算机能阅读的方式提供的一个正式的描述文档的语言。WSDL 基于 XML 语言，用于描述 Web 服务以及 Web 服务的函数、方法的参数和方法的返回值等信息。

WSDL 使用 XML 列举了某个 Web 服务的所有方法、所有参数以及方法的返回值。正因为其使用的是 XML，所以可以被计算机阅读，又可以被人阅读。

4．UDDI

UDDI(Universal Description, Discovery and Integration，通用描述、发现与集成服务)是一种

目录服务。企业使用它可以对 Web 服务进行注册和搜索。

UDDI 是一套基于 Web 的、分布式的、专为 Web 服务提供的、信息注册中心的实现标准规范。同时 UDDI 也包含了一组使企业能将自身提供的 Web 服务注册，以使其他企业能够发现和访问的协议实现标准。

为了使用 Web 服务，客户当然需要知道相应公司提供的 Web 站点地址或者发现文件的 URL。这个发现文件是非常有用的，它们可以将多个 Web 服务合并到一个单独的列表中。但是它们不允许客户在不了解这个公司的情况下搜索 Web 服务的信息。

UDDI 提供了一个数据库来填补这个缺陷，企业可以在这个数据库中发布自己的企业信息、提供的 Web 服务信息、第一个服务的类型，以及与这些服务有关的其他信息和规范的链接。

不过即使你在 UDDI 注册服务中找到了想要的 Web 服务，也只是能得到一个方法定义的集合，文档说明非常少，可以说相当于一个非常简单的 API 参考。

5．远程过程调用 RPC

Web 服务本身实现的是应用程序间的通信。通常情况下，我们在应用程序之间通信使用的是消息传递的方式。但是前面我们也已经看到了，在客户端使用 Web 服务的时候就像调用一个本地方法一样简单(如图 1-16 所示)，它是怎么实现的呢？也就是说它如何实现从消息传递到远程过程调用的方式呢？图 1-16 所示的是消息传递到过程调用的示意图。

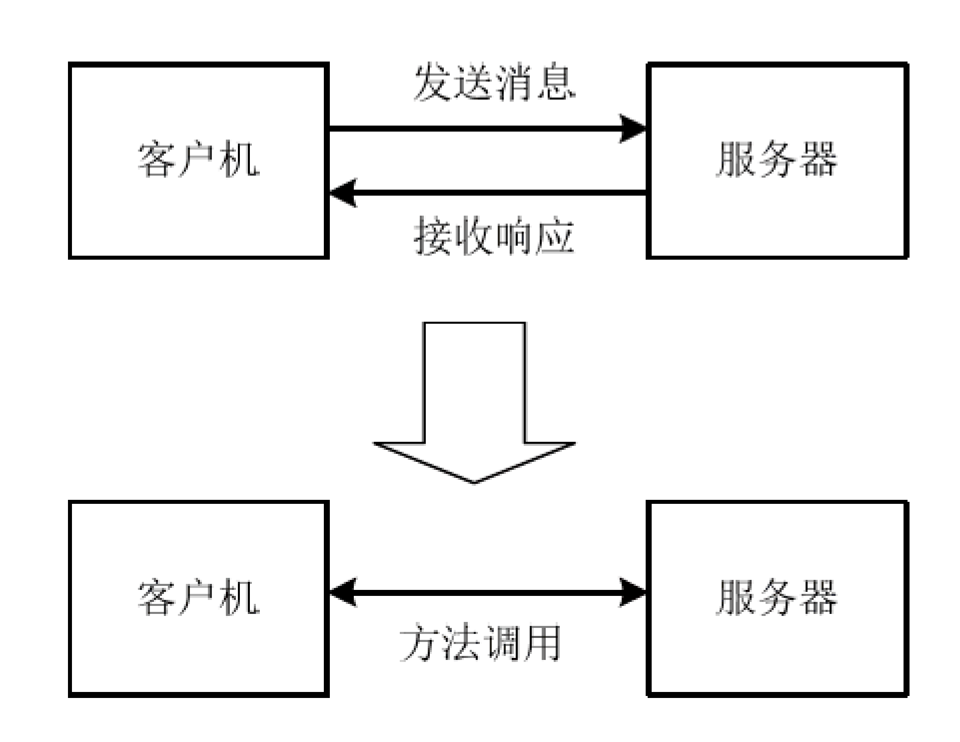


图 1-16　消息传递到远程过程调用

为了实现这种简便的远程过程调用的形式，Web 服务对这些信息进行了简单的映射，同样还使用消息传递的方式来实现 Web 服务的远程调用。也就是说 Web 服务的远程过程调用是靠内部的消息传递来完成的，如图 1-17 所示。

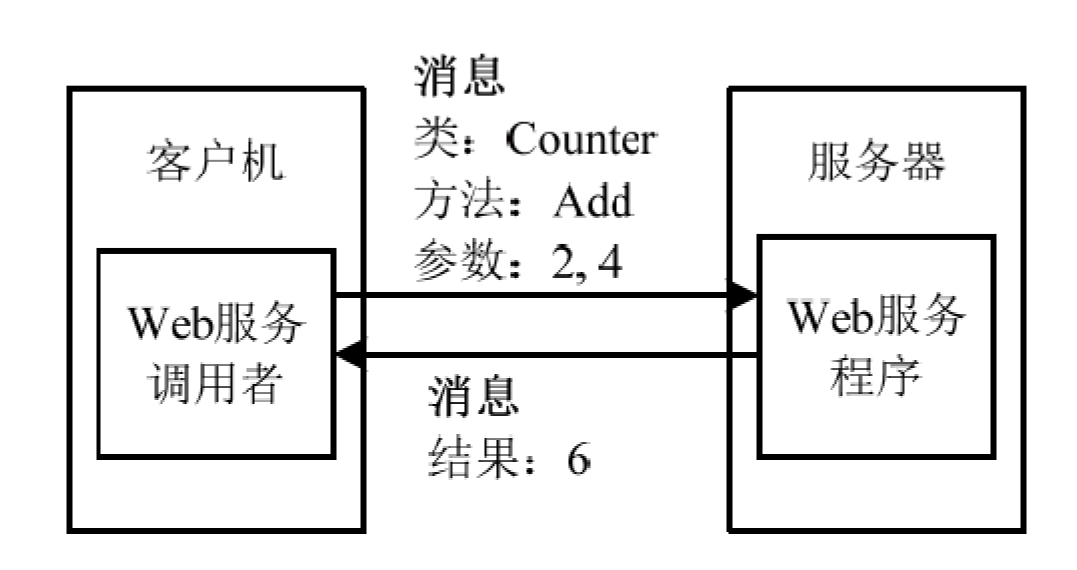


图 1-17　执行远程过程调用

远程过程调用是一种协议，全称为 Remote Procedure Call Protocol，简称 RPC。使用 RPC 的时候，客户端的运行方式是调用服务器上的远程过程。



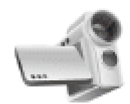
这里的“过程”相当于.NET 中的方法。在早些时候的编程语言中，没有“方法”这个概念，甚至还没有“函数”这个概念，所以称之为“过程”。

调用服务器上的远程过程，在.NET 中实现的通常方式是实例化一个远程对象并调用其方法或属性。

远程过程调用 RPC 倾向于使 Web 服务的位置透明化。服务器可提供远程对象的接口，客户端使用服务器中的远程方法就像在本地使用的这些 Web 服务对象的接口一样，这样就隐藏了 Web 服务的底层实现信息，客户端也不需要知道对象具体指向的是哪台主机。

1.5　使用 ASP.NET 构建一个 Web 服务

前面我们已经全面地了解了 Web 服务的原理，并且已经在.NET 中使用控制台程序实现了对 Web 服务的访问。当然，ASP.NET 作为一个优秀的开发技术，其中也提供了对 Web 服务的开发支持；不仅如此，使用 ASP.NET 开发 Web 服务程序是一件非常容易的事情。下面我们通过一个简单的实例，来演示一下 ASP.NET 中的 Web 服务。



视频教学：光盘/videos/01/构建一个 Web 服务.avi　　长度：6 分钟

1.5.1　实例描述

在 ASP.NET 中，使用集成开发环境 Visual Studio 创建一个 Web 服务的过程非常简单。我们不需要做任何配置，只需要根据需求编写相应的业务逻辑代码即可。

本实例，我们就来使用 Visual Studio 在 ASP.NET 应用程序中创建一个执行加法计算的 Web 服务。

1.5.2　实例应用

【例 1-4】 使用 ASP.NET 构建一个 Web 服务

(1) 首先需要创建一个“ASP.NET Web 应用程序”项目。

(2) 在该项目的根目录下创建一个“Web 服务”项目，命名为 Counter.asmx。创建成功以后系统会自动生成模板代码，其中包含一个示例方法，代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
namespace WebService
```

```
{
    /// <summary>
    /// Counter 的摘要说明
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // 若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
    // [System.Web.Script.Services.ScriptService]
    public class Counter : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

(3) 这里我们需要执行加法运算，所以我们来修改生成的 Web 方法 HelloWorld，修改为具有两个参数的 Add 方法，具体代码如下：

```
public class Counter : System.Web.Services.WebService
{
    [WebMethod]
    public decimal Add(decimal n1, decimal n2)
    {
        return n1 + n2;
    }
}
```

至此，我们的 Web 服务就算创建成功了。

(4) 接下来，使用一个访问 Web 项目的客户端程序。这里我们还使用一个控制台程序来访问这个 Web 服务。

(5) 为了使大家更容易理解，我们重新创建一个项目，用来调用该 Web 服务。这里我们创建一个控制台项目。

(6) 在新创建的控制台项目中添加一个服务引用，引用地址为刚才创建的 Web 服务的地址(这里是 http://localhost:7735/Counter.asmx)，并且在“命名空间”文本框中设置新创建的 Web 服务的命名空间为 ServiceCounter。

(7) 修改控制台项目中自动生成的 Program 类，添加使用 Web 服务执行加法操作的代码，如下所示：

```
class Program
{
    static void Main(string[] args)
    {
        ServiceCounter.CounterSoapClient counter =
            new ServiceCounter.CounterSoapClient();
        Console.Write("请输入加数：");
```

```
        decimal value1 = decimal.Parse(Console.ReadLine());
        Console.Write("请输入被加数: ");
        decimal value2 = decimal.Parse(Console.ReadLine());
        var result = counter.Add(value1, value2);
        Console.WriteLine(result);
        Console.ReadLine(); //程序暂停，等待用户回车确认
    }
}
```

如此，整个实例就算完成了，下面我们来测试一下。

1.5.3 运行结果

首先，需要运行 Web 项目。

接下来访问创建的 Web 服务的地址 http://localhost:7735/Counter.asmx，访问结果如图 1-18 所示。



图 1-18　Web 服务 Counter

可以看到，其界面和前面测试的其他 Internet 上发布的 Web 服务一模一样。这里我们单击 Add 超链接对该操作进行访问，结果如图 1-19 所示。



图 1-19　Web 服务 Counter 中的 Add 操作

打开 Add 操作的测试页面，下面同样有测试模块和一些简单的说明。在测试模块的 n1 文本框中输入 23.5，在 n2 文本框中输入 56，然后单击【调用】按钮，系统会请求 Web 服务器，并返回执行结果，如图 1-20 所示。

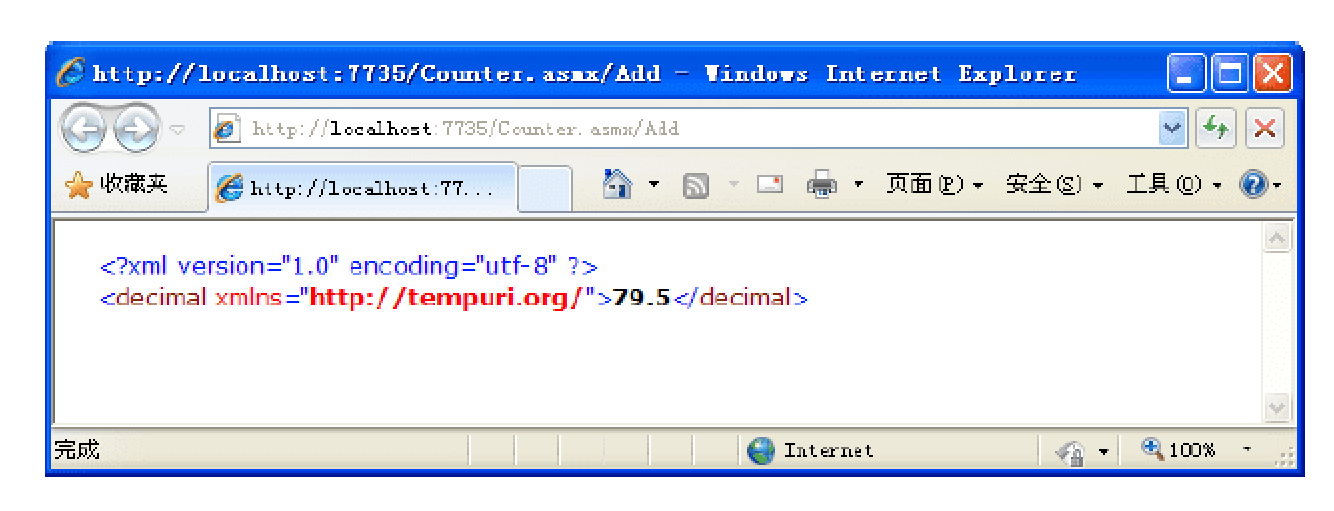


图 1-20　执行结果

然后再来测试控制台访问 Web 服务的结果。运行控制台程序，根据提示输入相应的数字，稍等即可输出响应结果，如图 1-21 所示。



图 1-21　控制台程序执行结果

1.5.4　实例分析


源码解析：

本实例，首先创建一个 ASP.NET Web 应用程序，在该 Web 应用程序中创建一个 Web 服务；然后系统会自动生成代码文件，其中包含一个简单的 Web 服务方法；接着将这个 Web 服务方法修改为一个执行加法操作的方法；最后创建一个控制台程序引用新创建的 Web 服务，并接收用户输入的数值，调用远程的方法执行加法操作，并打印出执行结果。

1.6　常见问题解答

1.6.1　什么是 Web Service



什么是 Web Service?

网络课堂：http://bbs.itzcn.com/thread-14984-1-1.html

今天听到有人在说 Web Service，Web Service 是什么呢？直接从字面上理解，是 Web 服务，但是 Web 服务又是什么呢？

【解决办法】

Web Service 是一种新的 Web 应用程序分支，它们是自包含、自描述、模块化的应用，可以发布、定位并通过 Web 调用。

Web Service 可以执行从简单的请求到复杂商务处理的任何功能。一旦部署以后，其他 Web Service 应用程序可以发现并调用它部署的服务。

Web Service 是一种应用程序，它可以使用标准的互联网协议，像超文本传输协议(HTTP)和 XML，将功能纲领性地体现在互联网和企业内部网上。可将 Web 服务视作 Web 上的组件编程。

1.6.2 Web Service 和 Web Server 的区别



Web Service 和 Web Server 的区别。
网络课堂：http://bbs.itzcn.com/thread-14985-1-1.html

今天听到有人在讲 Web Service，我记得以前上学的时候学到过一个 Web Server 的概念。它们有什么区别呢？

【解决办法】

概念上根本就是两个东西。像 IIS、Apache 这类的软件都叫做 Web Server，可以让用户通过用 IE 访问服务器的 IP 看到一个页面，或者完成从页面上传递来的数据交互。而 Web Service 仅仅是一个 API。

比如要想创建一个 Web Service，它的作用是返回当前的天气情况，那么可以建立一个 Web 页面，用于接受邮政编码作为查询字符串，然后返回一个由逗号隔开的字符串，包含当前的气温和天气，要调用这个 Web 页面，并向它传入参数，就可以返回这样格式的一行数据：“21,晴”。那么这个 Web 页面就应该可以算作是一个 Web Service 了。因为它基于 HTTP GET 请求，提供了一个可以通过 Web 调用的 API。

当然，真正的 Web Service 肯定不是这么实现的，不过内部原理都一样。

1.7 习　　题

一、填空题

(1) Web 服务是指在网络中发布的一些__________，可以供网络中其他的应用程序访问。

(2) __________是为 Web 服务所选择的消息传递协议。

(3) WSDL 基于__________，用于描述 Web 服务、Web 服务的函数、方法的参数和方法的返回值等信息。

二、选择题

(1) 下面__________不是使用 Web 服务的优点。

A. Web 服务是一种优秀的分布式计算技术

B. Web 服务可以轻松地跨过防火墙

C. Web 服务使用的 SOAP 协议非常简单

D. Web 服务使数据更加分散，保证了数据的安全性

(2) 下面选项中，__________不是 Web 服务用到的技术。

A. XML　　B. SOAP　　C. WSDL　　D. SMTP

(3) 下面所有使用 Web 服务的环境，__________使用得非常不恰当。

A. 银行提供的网上支付接口

B. 货运公司提供的查询运单信息的功能

C. 企业网站的产品管理功能

D. 企业 HR 系统中的员工信息功能

三、上机练习

上机练习 1： 使用 Web 访问 Internet 中提供的免费天气预报功能。

在本章中我们已经对 Web 服务进行了比较深入的介绍。也使用网络中提供的一些商业 Web 服务进行了一些简单演示。

本练习，我们就在 Internet 中找一个提供免费天气预报功能的 Web 服务，然后在浏览器中查询当前最新的天气预报信息，如图 1-22 所示。

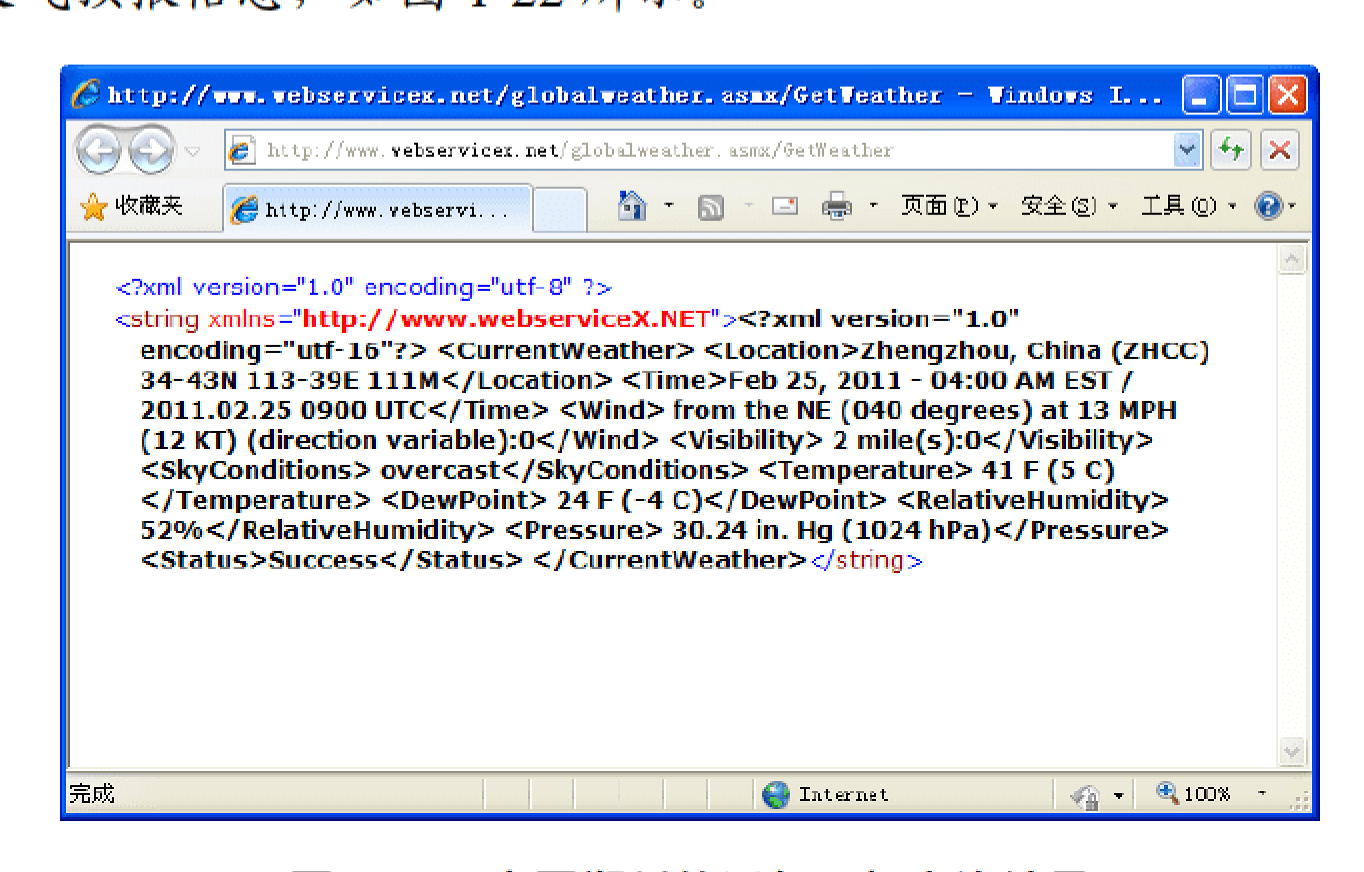


图 1-22　中国郑州的天气预报查询结果



第 2 章　构建 ASP.NET Web 服务

内容摘要：

通过第 1 章的学习，我们了解了 Web 服务的相关技术和架构，还对 Web 服务的使用有了简单认识。

我们知道，Web 服务实际上是一种结构、一种组织应用软件的模式。它并没有基于任何一种开发工具、语言甚至运行平台。但是，作为 Web 服务的开发者，必须选择一种自己熟悉的、效率高的开发工具和编程语言进行开发。

在这方面，ASP.NET 无疑是最佳选择。它采用的.NET Framework 框架可以很容易地创建和运行一个 Web 服务，而且借助于 Visual Studio 开发工具的支持能够大大减少开发人员的工作。

在本章中，我们将学习创建 Web 服务的各种方法，并重点对使用 Visual Studio 创建 ASP.NET Web 服务和修改 Web 服务的属性进行介绍。

学习目标：

- 掌握如何用记事本创建 Web 服务
- 了解 Web 服务处理指令的用法
- 掌握 Web 服务类和方法的使用
- 掌握 wsdl 和 csc 命令的使用
- 掌握 Visual Studio 创建和测试 Web 服务的方法
- 了解添加服务引用与 Web 引用的区别
- 熟悉创建 Web 服务时的 WebService 属性
- 了解如何使用 WebMethod 属性控制 Web 服务方法

2.1　使用记事本创建 Web 服务

虽然 Visual Studio 为.NET 开发提供了功能丰富的集成开发环境，但是这并不是创建 Web 服务的唯一途径。使用一种文本编辑器(例如记事本)和.NET Framework 包含的命令行工具，也可以创建 Web 服务。

**视频教学：**光盘/videos/02/记事本创建 Web 服务.avi　　　　长度：13 分钟

2.1.1　基础知识——WebService 处理指令

由于使用记事本编写的程序都为纯文本内容，为了使其能够被正确识别为 Web 服务，首先需要以.asmx 为扩展名进行保存。

此外，为了将文件中的类和方法被编译器标识为 Web 服务，还需要一个特殊的 ASP.NET 处理指令——WebService。该处理指令必须写在文件的第一行，用于表明实现 Web 服务的类，以及使用的编程语言等属性。

WebService 处理指令的使用示例如下：

```
<%@ WebService Language="C#" Class="TestWebService " %>
```

要想使 Web 服务正常工作，WebService 处理指令的 Language 和 Class 属性都必须设置。其中，设置 Language 属性可以告诉.NET Framework 该 Web 服务中的类是使用哪一种程序设计语言编写的；设置 Class 属性是为了告诉 ASP.NET 将要作为 Web 服务表述的类的名称。

在这个例子中，WebService 处理指令指定 Web 服务的类为 TestWebService，使用的编写语言为 C#。这实际上将通知.NET Framework 需要使用 C#编译器把类 TestWebService 编译成一个具有 Web 输出方法的 Web 服务。



WebService 处理指令还有一个 Codebehind 属性是可选的，用来指定当前 Web 服务所用的后台代码文件。

2.1.2　基础知识——声明 Web 服务类和方法

声明 Web 服务类与声明其他普通类没有任何区别，例如在上节声明的 TestWebService 类。不过，如果想要在 Web 服务中访问 ASP.NET 的内置对象，像 Application、Session 和 Context 等，则该 Web 服务器必须继承自 WebService 类。

另外，在实现 Web 服务的类中可能会有很多方法，但并不是所有的方法都可以通过 Web 进行调用。而是只有那些添加了 WebMethod 属性的方法才可以调用，而且必须声明为 public。

通常，我们也将在 Web 服务类中带有 WebMethod 属性，且为 public 类型的方法简称为 Web 服务方法。

例如，下面的示例代码创建了一个 Web 服务类 TestWebService，并提供了一个名为 Hello

的 Web 服务方法。

```
using System.Web.Services;
public class TestWebService:WebServices
{
    [WebMethod]
    public string Hello()
    {
        return "Hello";
    }
}
```

2.1.3　实例描述

Visual Studio 对 Web 服务提供了强大的支持，强大到我们都不知道它干了些什么。

没关系，本节我们先不用工具，而用记事本创建一个简单的 Web 服务。从本质上了解一下 Web 服务的组成及创建方法。

别小看这节啊。只有吃尽过去的苦，才明白现在的幸福。同样你只有深刻地体会了不用 Visual Studio 的痛苦，才能知道使用 Visual Studio 的幸福。

2.1.4　实例应用

【例 2-1】 使用记事本创建 Web 服务

(1) 打开 Windows 系统自带的记事本程序，添加如下的代码：

```
<%@ WebService Language="C#" Class="myFirstWebService" %>
using System.Web.Services;
public class myFirstWebService : WebService
{
    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World!";
    }
    [WebMethod]
    public string Hello(string name)
    {
        return "欢迎" + name + "的访问，当前时间为：" + System.DateTime.Now;
    }
}
```

(2) 将代码保存到一个文件中。注意，保存的时候应该以.asmx 为扩展名。例如，在这里我们将文件保存为 myFirstWebService.asmx。

(3) 这样一个 Web 服务就创建好了。为了使其能够通过浏览器访问，我们在 IIS 上新建一个虚拟目录 firstService，它指向本地磁盘的物理路径(F:\myfirst)。

(4) 将 Web 服务文件 myFirstWebService.asmx 复制到新建的 firstService 目录下，完成实例。

2.1.5 运行结果

现在，我们需要在 IIS 下运行新建的 Web 服务。很简单，就像运行 ASP.NET 页面一样，在浏览器中输入 Web 服务的 URL 来打开 myFirstWebService.asmx 文件就行了。

在本实例中，我们输入“http://localhost/firstService/myFirstWebService.asmx”之后，Web 服务的浏览页面如图 2-1 所示。



图 2-1 浏览 Web 服务



如果读者是先安装.NET，再安装 IIS，浏览.asmx 文件时可能会弹出“访问 IIS 元数据库失败”的错误。解决办法是，在命令提示符下进入.NET Framework 的安装目录，例如“C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727”，再运行“aspnet_regiis.exe -i”命令修复 IIS 即可。

如图 2-1 所示，可以看到页面开始显示的是 Web 服务名称，再往下是 Web 服务中可调用的方法：Hello 和 HelloWorld，再往下的内容是如何为 Web 服务器定义命名空间等信息。

单击其中一个方法的链接，则会出现一个用于测试该方法的测试页面。如果该方法有参数，那该测试页面将会有用于输入 Web 测试参数的文本输入框。

例如，单击 Hello 链接，进入的调用页面如图 2-2 所示。

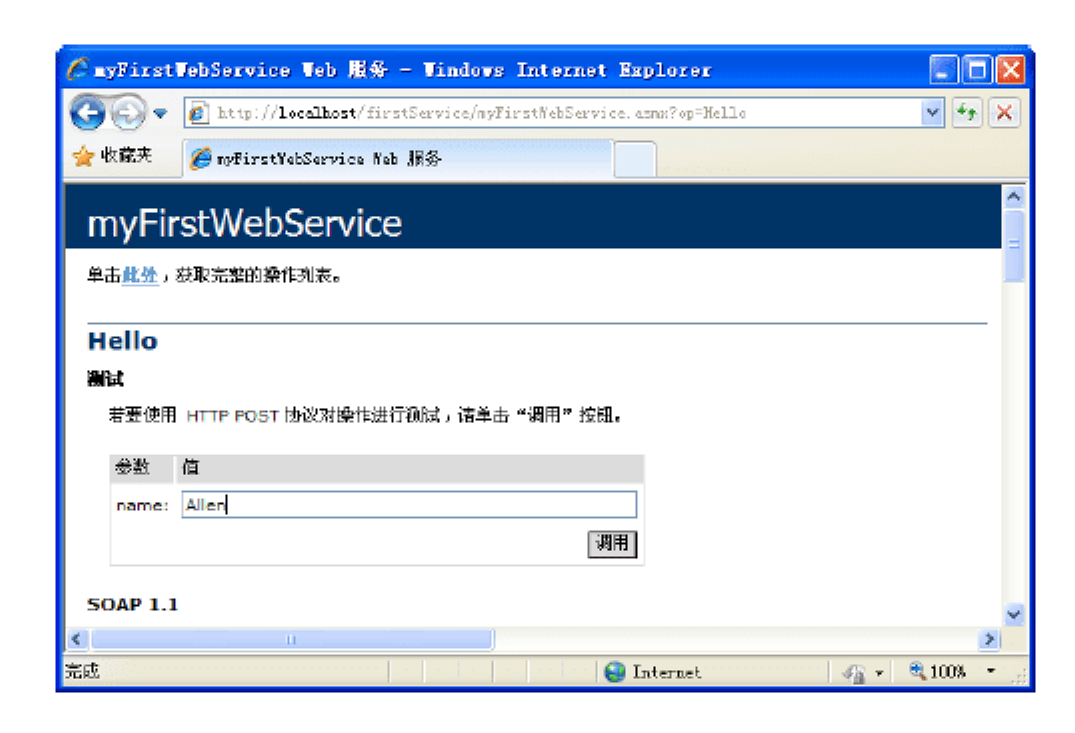


图 2-2 调用 Hello 方法

在这里为 name 参数指定一个值，再单击【调用】按钮测试该 Web 服务方法，结果将以 XML 形式显示，如图 2-3 所示。

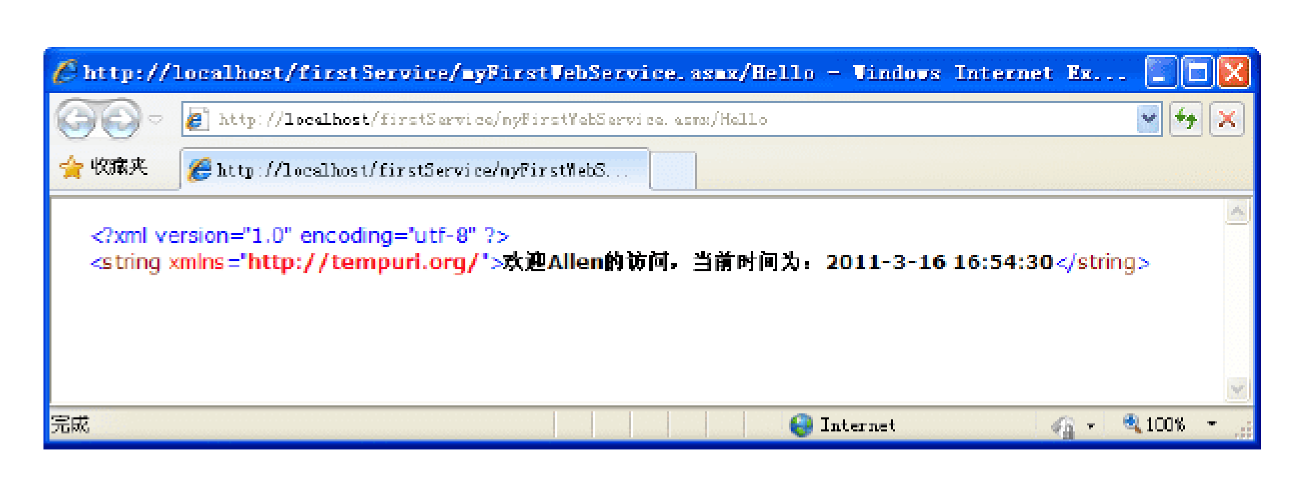


图 2-3　调用 Hello 方法的显示结果

如果出现了 XML 文档格式的页面，那么恭喜你，Web 服务创建成功了。

2.1.6　实例分析


源码解析：

在该实例中，我们用记事本编写 Web 服务，从最简单的实例中了解 Web 服务的组成和创建方法。需要注意的是，在保存的时候需要将记事本保存为.asmx 格式的文件，因为.asmx 文件是 Web 服务的入口。并且需要将文件保存到 IIS 虚拟目录对应在本地磁盘的物理路径下。

如果需要将方法添加到 Web 服务里，需要向该方法添加 WebMethod 属性。

2.2　从命令行执行 Web 服务

使用 Web 服务的过程实际上就是将 Web 服务的使用者与 Web 服务实现绑定，然后再调用其中的方法。对于绑定最简单、最常用的方法就是创建一个 Web 服务的 WSDL。本节，我们仍然从命令行形式进行讲解如何绑定、编译和执行一个 Web 服务。


视频教学：光盘/videos/02/命令行执行 Web 服务.avi　　长度：6 分钟

2.2.1　基础知识——创建代理类

.NET 客户端用户应用程序可以通过一个代理与 Web 服务进行通信。这个代理是从 Web 服务 WSDL 文档创建的一个.NET 程序集。既可以由 Visual Studio 自动创建 Web 服务代理，也可以使用由.NET Framework 提供的 wsdl.exe 创建代理类源代码，然后再使用.NET Framework 中的 csc.exe 编译器将代理类编译为代理程序集(.dll 文件)。

创建代理之后，Web 服务使用者只需从中调用 Web 方法，代理将会执行 Web 服务的实际请求，在请求中也可以指定端点的位置，无论是在局域网内还是通过 Internet 连接到另一个国家或洲都可以。当用户在使用者应用程序中引用 Web 服务时，它就如同是使用者应用程序的一部分，与通常的内部函数调用一样。图 2-4 就演示了这一过程。

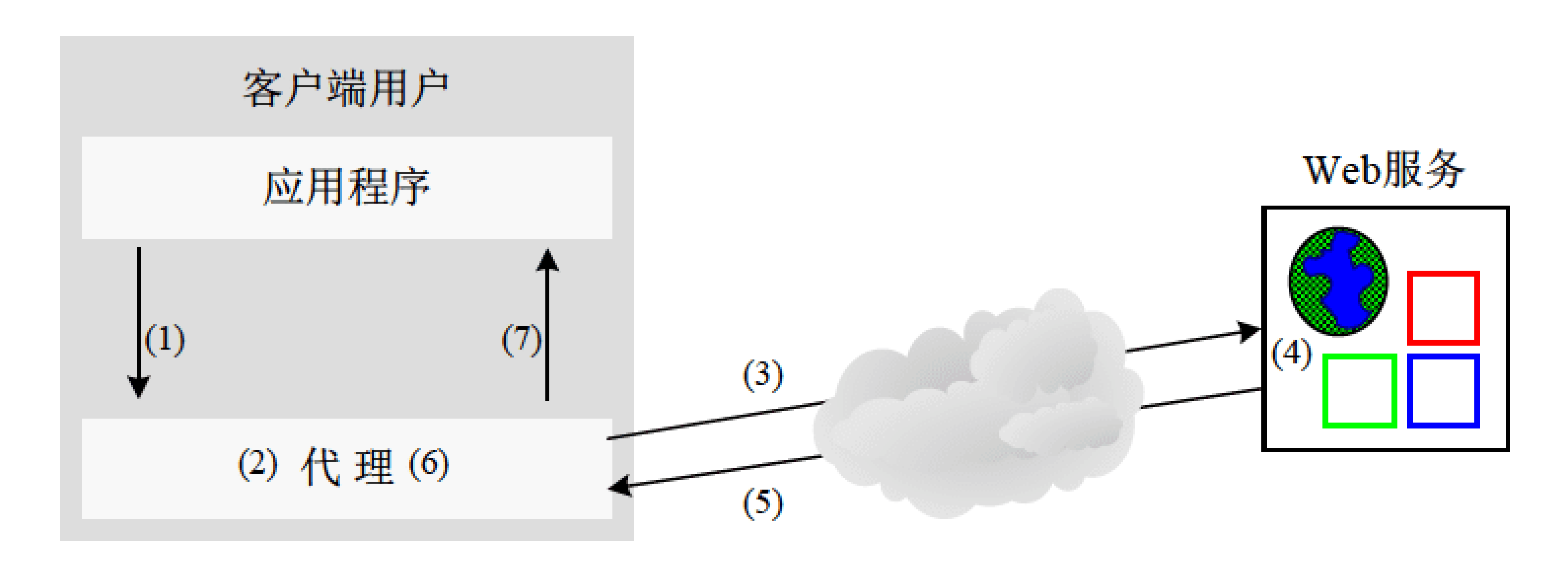


图 2-4　代理使用 Web 服务的过程

这一过程包含以下几个步骤。

(1) 应用程序执行代理代码中的函数，为其传递合适的参数，但并不知道代理将要调用 Web 服务。

(2) 代理接收到调用后，格式化将要发送到 Web 服务的请求，该 Web 服务是使用者用参数指定的。

(3) 函数调用从代理发送到 Web 服务。调用可以发生在同一台计算机之中，也可以通过局域网或者 Internet 进行调用。

(4) Web 服务使用代理提供的参数执行其可以通过 Web 调用的函数，并把结果放入 XML 文档中。

(5) 来自 Web 服务的结果数据返回给使用者处的代理。

(6) 代理分析来自 Web 服务的 XML，获取生成的各个值。

(7) 使用者应用程序从代理函数获得需要的值，完全不必考虑它们来自 Web 服务调用。

这种代理方法的好处是将.NET 用户应用程序与在代理中处理的分布式调用过程分隔开。这意味着如果基本的消息格式或者传输协议发生了变化，那么只需要重新编译代理，以使用新的消息或者传输协议，而不必重新编写使用者应用程序(当然，如果基本的通信语义发生了重大变化，通常需要重新编写应用程序)。从使用者应用程序的角度看，代理只不过是展示远程服务接口的另一个程序集。当然，虽然代理使得使用 Web 服务的过程变得更加容易，但是不使用代理也可以使用 Web 服务。

2.2.2　基础知识——使用命令生成代理

在使用命令生成代理前，我们要确保系统里安装了必备执行工具，它们是 wsdl.exe、csc.exe(如果采用 VB.NET，可能还需要 vbc.exe)，并且将它们的路径添加到系统变量里。

所以，我们要对环境变量进行配置，只有这样，才能保证在任何目录下进行操作。对于 Visual Studio 2010 来说，各路径如下：

```
csc.exe 位于 C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319
wsdl.exe 位于 C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin
```

添加方法为：右击【我的电脑】图标，在弹出的快捷菜单中选择【属性】命令，在弹出的对话框中打开【高级】选项卡；再单击【环境变量】按钮从弹出的对话框中双击 Path 变量；之后将上面的路径添加到【变量值】文本框中即可，两个路径之间用分号分隔，如图 2-5 所示。

图 2-5　编辑环境变量

接下来，测试环境变量是否生效。进入命令提示符窗口，输入 WSDL，将会得到该命令的帮助，如图 2-6 所示，则说明添加成功。

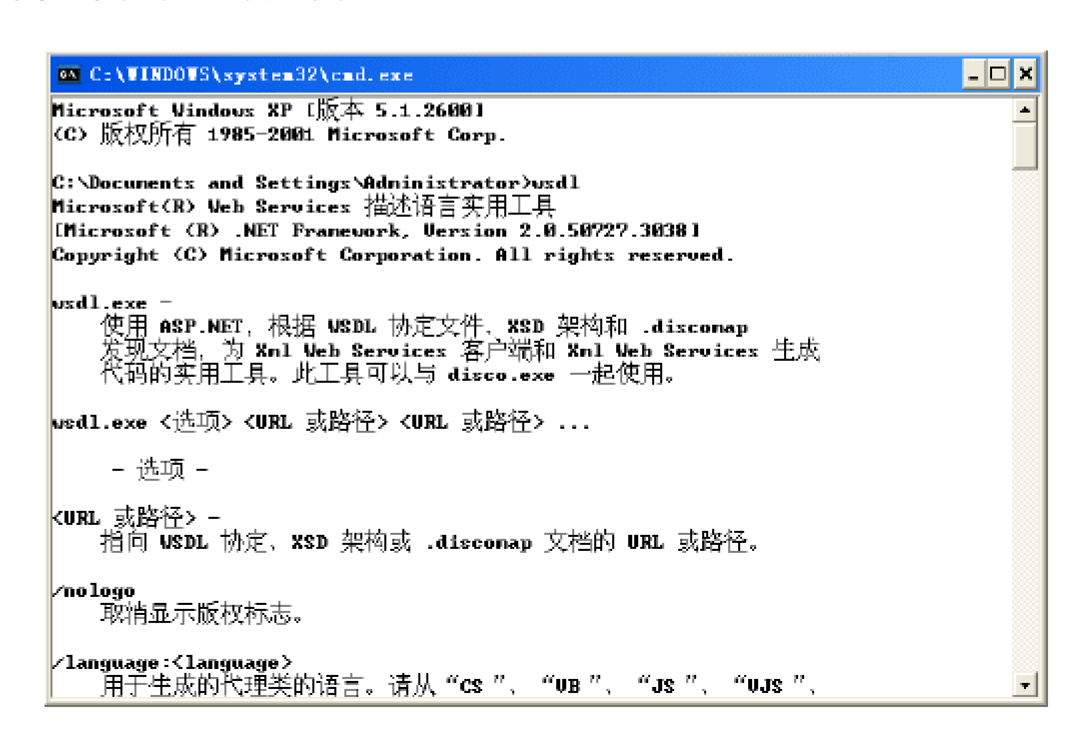


图 2-6　测试 WSDL

wsdl.exe 工具是.NET Framework 的一部分，只需指定 Web 服务描述文档的 URL，wsdl.exe 就会自动生成可以编译为代理 DLL 的源代码。wsdl.exe 工具的命令行语法如下：

```
wsdl.exe  < 参数 > < URL 或者 路径 >
```

在表 2-1 中列出了 wsdl.exe 命令的常用参数及说明，第二个参数为通过 URL 或者路径方式指定一个要生成代理的 Web 服务。

表 2-1　wsdl.exe 命令参数

参　数	说　明
/u:	指定需要认证的服务器的用户名
/p:	需要认证的服务器的密码选项
/o:	指定输出源代码的文件名。默认值是根据 Web 服务的名称生成的
/n:	指定输出代理代码的命名空间
/l:	指定代理类源代码所使用的语言，可选的语言选项有 CS(C#，默认)、VB 和 JS
/protocol:	指定代理类将用于通信的协议，代理类可以在多种协议如 SOAP(默认)、HttpGet 或者 HttpPost 上运行；也可以指定 Web 服务支持的自定义协议

在为 Web 服务创建代理类之后，还需要将代理类源代码编译为一个程序集，这可以使用 csc.exe 编译器来实现。表 2-2 中列出了 csc.exe 编译程序的部分命令行参数及其说明。

表 2-2　csc.exe 命令参数

参　数	说　明
/out:<文件>	指定输出文件的名称和位置，默认时，编译器从源文件名得到输出名称
/t:	指定输出文件的类型(包括 exe、winexe、library、module 等类型)
/r:	指定一个程序集列表，这些程序集以逗号或者分号隔开，它们可以在编译时使用
/doc:<文件>	指定要生成的 XML 文档文件
/checked[+|-]	是否启用生成溢出检查
/unsafe[+|-]	是否允许“不安全”代码
/nologo	取消编译器的版权信息
/utf8output	指定以 UTF-8 编码格式输出编译器信息

2.2.3　实例描述

无论是否使用 Visual Studio 开发工具，我们都能够很容易地开发 Web 服务。而且在上节中，我们已经掌握了如何用记事本创建一个 Web 服务并在 IIS 上测试。

下面我们同样不需要借助工具，而是要进入命令提示符窗口来完成 Web 服务代理的创建，通过执行命令来编译，并编写控制台程序进行测试。

2.2.4　实例应用

【例 2-2】 从命令行执行 Web 服务

(1) 上节我们创建了一个 myFirstWebService.asmx，并放到了 IIS 的 firstService 虚拟目录下。

(2) 进入命令提示符窗口，使用 wsdl 命令创建对 myFirstWebService.asmx 的代理。执行命令如下：

```
wsdl /l:cs /n:aSpace http://localhost/firstService/myFirstWebService.asmx
```

(3) 执行后，我们可以看到如图 2-7 所示的结果。

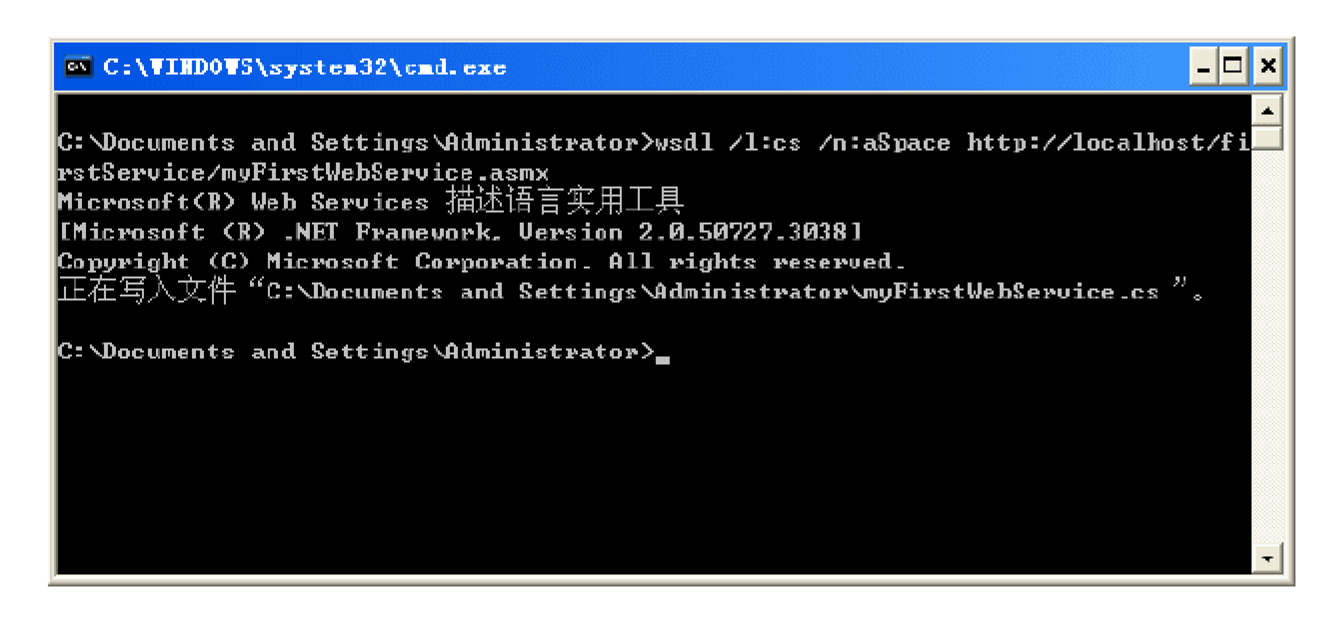


图 2-7　执行 wsdl 命令创建代理

根据提示，可以在 C:\Documents and Settings\Administrator 目录下看到生成了一个名为 myFirstWebService.cs 的文件。

(4) 创建好 C#文件后，需要把它编译成 DLL 文件，方便其他程序进行调用。这就需要用到 csc 命令，执行命令如下：

```
csc /out:GreetingsPS.dll /t:library /r:system.data.dll /r:system.web.services.
dll myFirstWebService.cs
```

执行后的结果如图 2-8 所示。

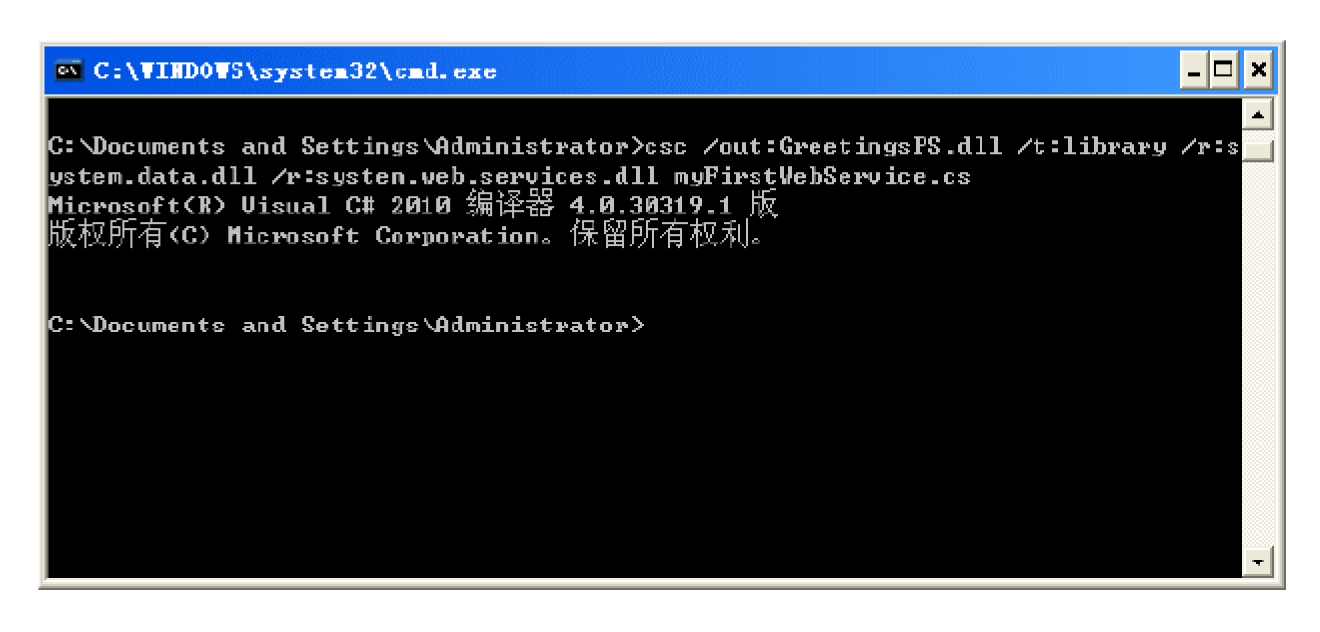


图 2-8　执行 csc 命令生成 DLL 文件

从结果中，可以看到生成了一个名为 GreetingsPS.dll 的 DLL 文件。这样，我们的代理就创建好了，如果其他程序想进行调用，那么只需要引用 GreetingsPS.dll 文件就可以了。

(5) 使用 Visual Studio 创建一个控制台应用程序，测试上面的 Web 服务是否正常。

(6) 在【解决方案资源管理器】窗格中打开【添加引用】对话框，然后通过【浏览】选项卡找到 csc 生成的 DLL 程序集文件，如图 2-9 所示。

(7) 切换到.NET 选项卡，找到 System.Web.Services 并单击，如图 2-10 所示。

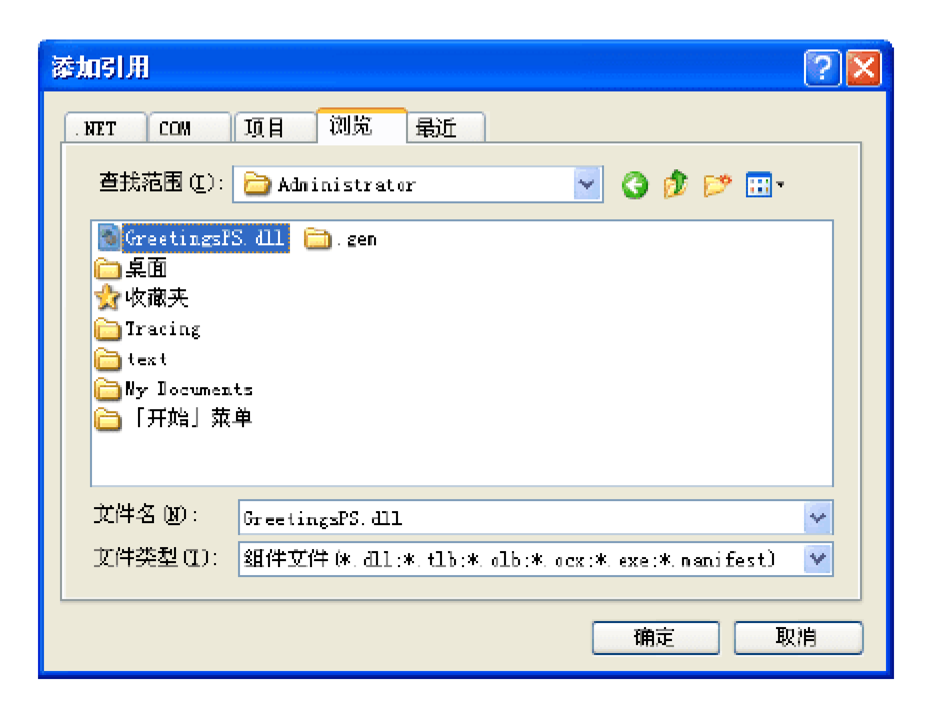


图 2-9　引用 GreetingsPS.dll 程序集



图 2-10　引用 System.Web.Services 程序集

(8) 单击【确定】按钮关闭对话框，此时引用就完成了。编写调用 Web 服务的代码，如下所示为最终文件的实现代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using aSpace;
```

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            myFirstWebService m = new myFirstWebService();
            string name = "王微微";
            Console.WriteLine();
            Console.WriteLine("调用 Web 服务后返回的信息为: " + m.Hello(name));
        }
    }
}
```

(9) 保存对文件的修改，完成实例的制作。

2.2.5 运行结果

按下 Ctrl+F5 组合键运行当前的控制台项目，将会在命令提示符下看到输出结果，如图 2-11 所示，说明调用成功。

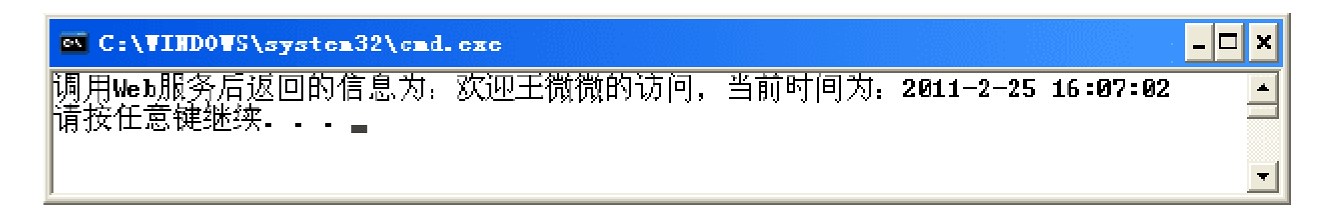


图 2-11 运行结果

2.2.6 实例分析



源码解析：

在使用命令生成代理中，我们首先要注意的是在环境变量中编辑系统变量 Path 的值，将 wsdl.exe 和 csc.exe 实用程序所在目录的路径写入系统变量 Path 的值字符串中。这样，在使用命令生成代理时，用户在命令行提示下就不用输入每个文件的完整目录了。

在引用的时候，我们不光要引用 csc 生成的 DLL 程序集文件，同时还要引用 System.Web.Services 程序集，否则程序无法运行。

2.3 实现用户登录验证的 Web 服务

在前面两节介绍了如何使用记事本和命令行编译器来创建 Web 服务。使用这种方式在编写简单的例子时还可以应付，当要编写由很多文件组成的大型项目时就显得力不从心了。

此时，我们便可以利用 Visual Studio 工具来创建 Web 服务。Visual Studio 可以将我们从复

杂的命令行中解脱，将主要精力集中到 Web 服务的实现上。

**视频教学：**光盘/videos/02/登录验证的 Web 服务.avi　　长度：9 分钟

2.3.1　基础知识——利用 Visual Studio 创建 Web 服务

和创建其他类型的应用程序一样，Visual Studio 为创建 Web 服务也提供了模板。我们所要做的工作只是在这个模板中添加所需的代码，非常方便。

本书采用的是 Visual Studio 2010，它是一个功能非常强大的集成开发环境，而且提供了对最新.NET Framework 4 的支持。此外，Visual Studio 2010 还提供了.NET Framework 3.5、.NET Framework 3.0 以及.NET Framework 2.0 版本下应用程序的模板。

但是要注意，.NET Framework 4 以后将使用 WCF 项目来代替之前的 Web 服务，它是对 Web 服务的优化和升级。如图 2-12 所示为创建基于.NET Framework 4 版本应用程序时的模板。

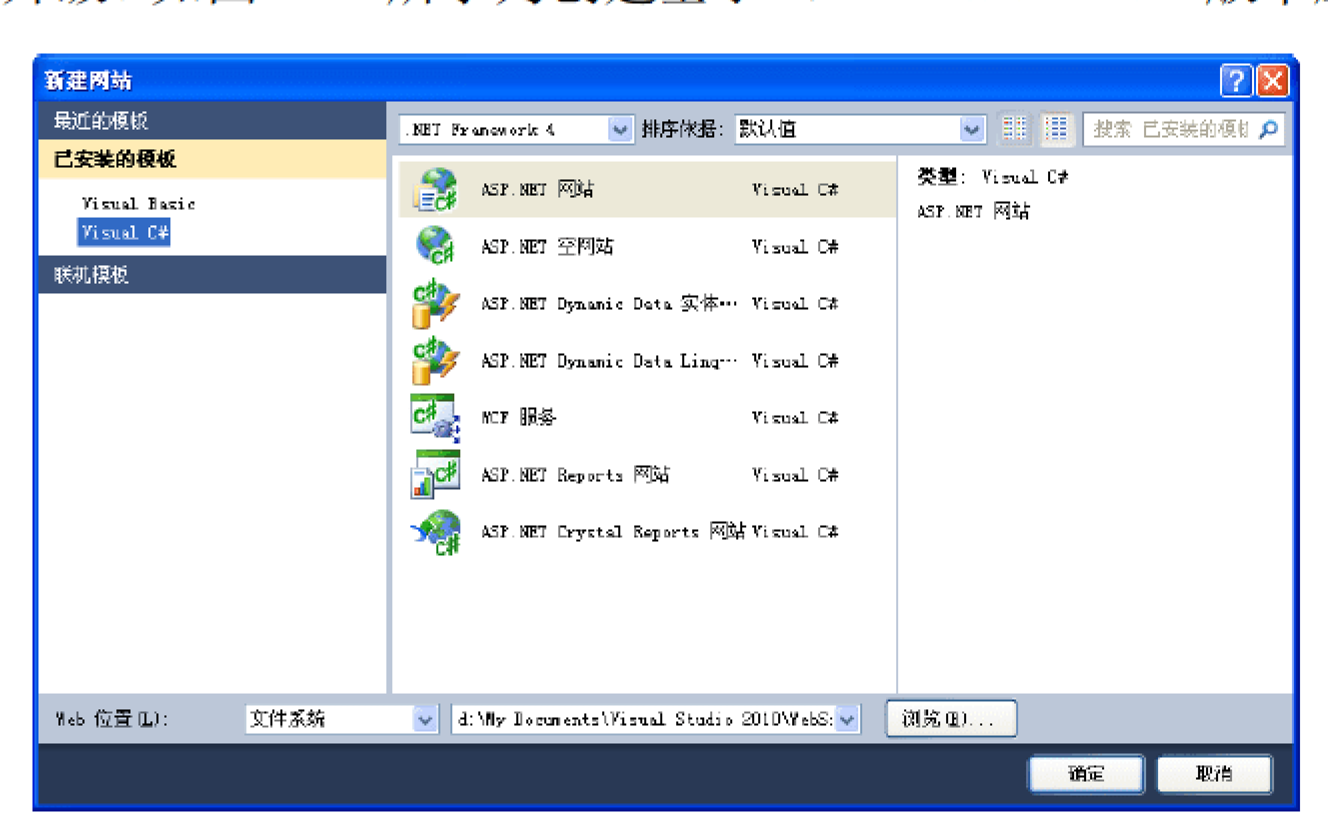


图 2-12　.NET Framework 4 版本下的模板

在.NET Framework 4.0 中没有提供与 Web 服务相关的模板。而.NET Framework 3.5 不仅提供了 Web 服务模板，还提供了 WCF 服务模板。

如图 2-13 所示为切换到.NET Framework 3.5 时看到的“ASP.NET Web 服务”模板。



图 2-13　.NET Framework 3.5 版本下的模板

2.3.2　实例描述

既然使用记事本等简单的文本编辑器创建和使用 Web 服务非常容易，那么为什么还要使用 Visual Studio 呢？其实答案很简单，Visual Studio 已经不仅仅是一种开发工具，而是一个全面的集成开发平台。

利用 Visual Studio 为用户提供的平台，我们可以开发、调试、测试、部署 Web 服务，而且可以通过“智能提示”功能来快速完成编码工作。

下面使用 Visual Studio 2010 创建一个 Web 服务实现用户登录验证的功能。

2.3.3　实例应用

【例 2-3】 利用 Visual Studio 创建 Web 服务

(1) 打开 Visual Studio 2010 的【新建网站】对话框，从顶部的下拉列表框中选择.NET Framework 3.5，【模板】选择“ASP.NET Web 服务”，更改【Web 位置】为“HTTP”，再输入一个当前网站要保存的 URL，这里为“http://localhost/UserWebService”，最后单击【确定】按钮，如图 2-14 所示。

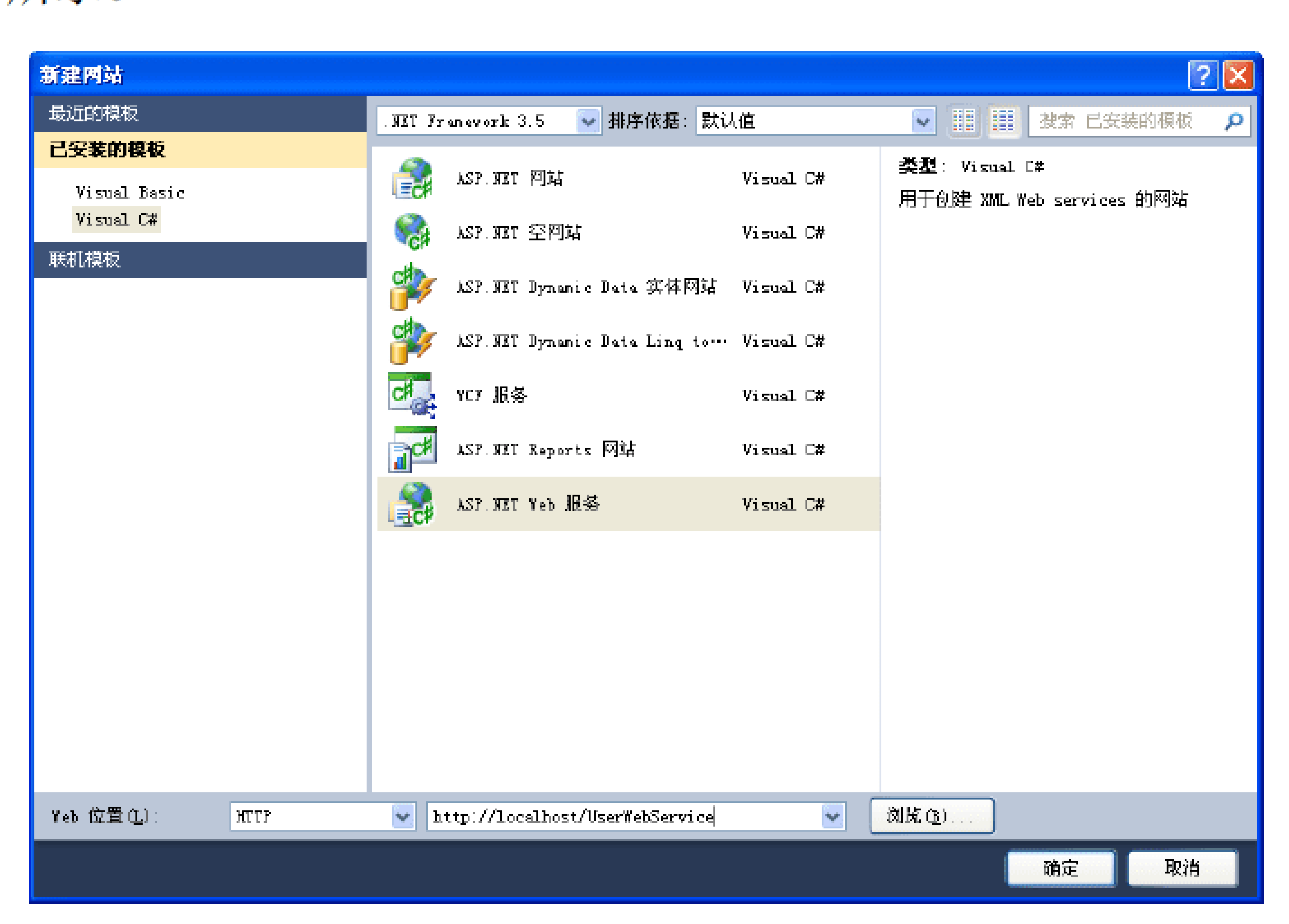


图 2-14　使用 Visual Studio 2010 创建 Web 服务



在这里如果选择.NET Framework 4，将看不到 ASP.NET Web 服务模板。

(2) 这样，我们的 Web 服务网站就已经创建好了。通过【解决方案资源管理器】窗格可以看到网站中包含的文件和文件夹，如图 2-15 所示。

Visual Studio 2010 自动生成的名为 Service 的 Web 服务中，包含两个文件：Service.asmx 和 Service.cs。这是因为在默认情况下，Visual Studio 2010 是使用“后台编码”形式创建 Web 服务的。

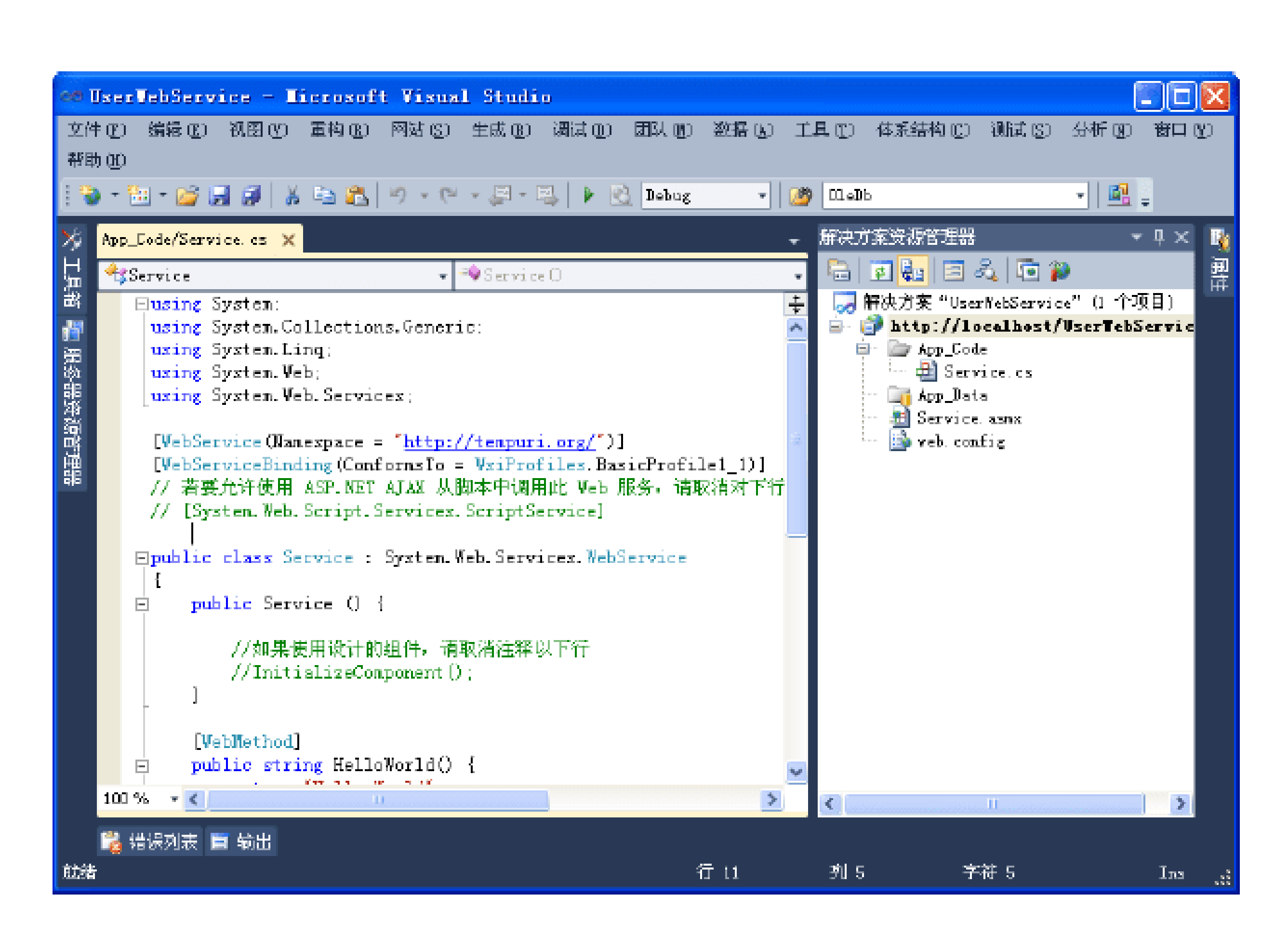


图 2-15　创建好的 Web 服务

(3) 右击网站名称并在弹出的快捷菜单中选择【添加新项】命令，在弹出的【添加新项】对话框中选择【Web 服务】模板，然后设置名称为“UserService.asmx”，再单击【添加】按钮，如图 2-16 所示。

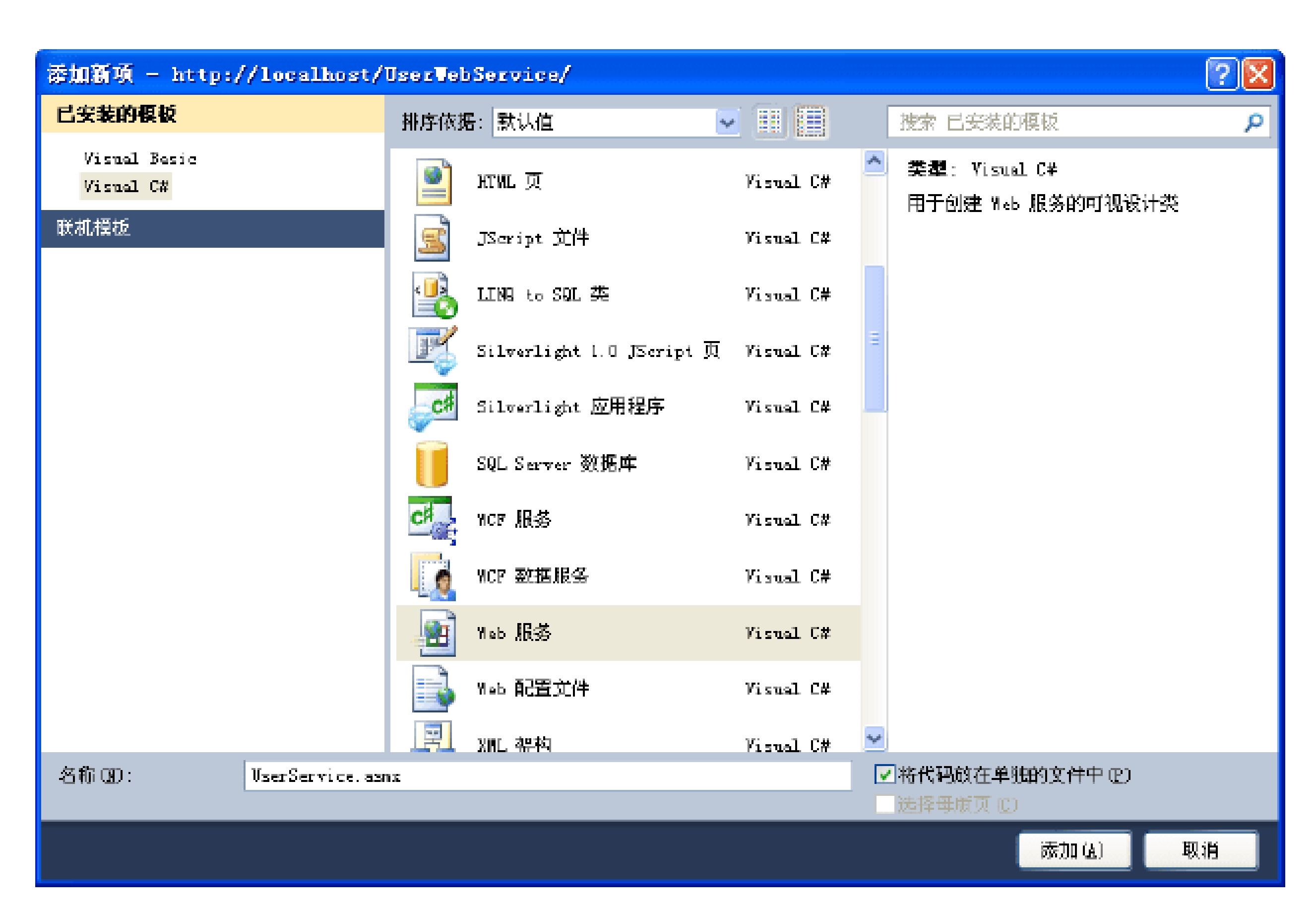


图 2-16　新建 Web 服务

(4) 打开新建的 Web 服务文件 UserService.asmx，可以发现文件里只有一行代码。

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/UserService.cs" Class="
UserService" %>
```

(5) CodeBehind 属性指定了与 UserService.asmx 文件相匹配的源代码文件是 UserService.cs。打开该文件，可以看到自动生成了如下代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
/// <summary>
```

```
///UserService 的摘要说明
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
//若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
// [System.Web.Script.Services.ScriptService]
public class UserService : System.Web.Services.WebService {
    public UserService () {
        //如果使用设计的组件，请取消下行的注释
        //InitializeComponent();
    }
    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}
```

(6) 接下来添加一个用于验证用户的方法 CheckLogin()，它的实现代码如下。在这里为 Web 服务方法加上 WebMethod 属性是必需的，否则无法调用该方法。

```
[WebMethod(Description="验证用户输入的用户名和密码是否正确，返回一个布尔值。")]
public bool CheckLogin(string name,string password)
{
    if (name == "admin" && password == "123456")
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

(7) 保存对文件的修改，完成实例的制作。

2.3.4 运行结果

现在，通过浏览器测试创建的 Web 服务 UserService。可以在浏览器中输入“http://localhost/UserWebService/UserService.asmx”，如图 2-17 所示。

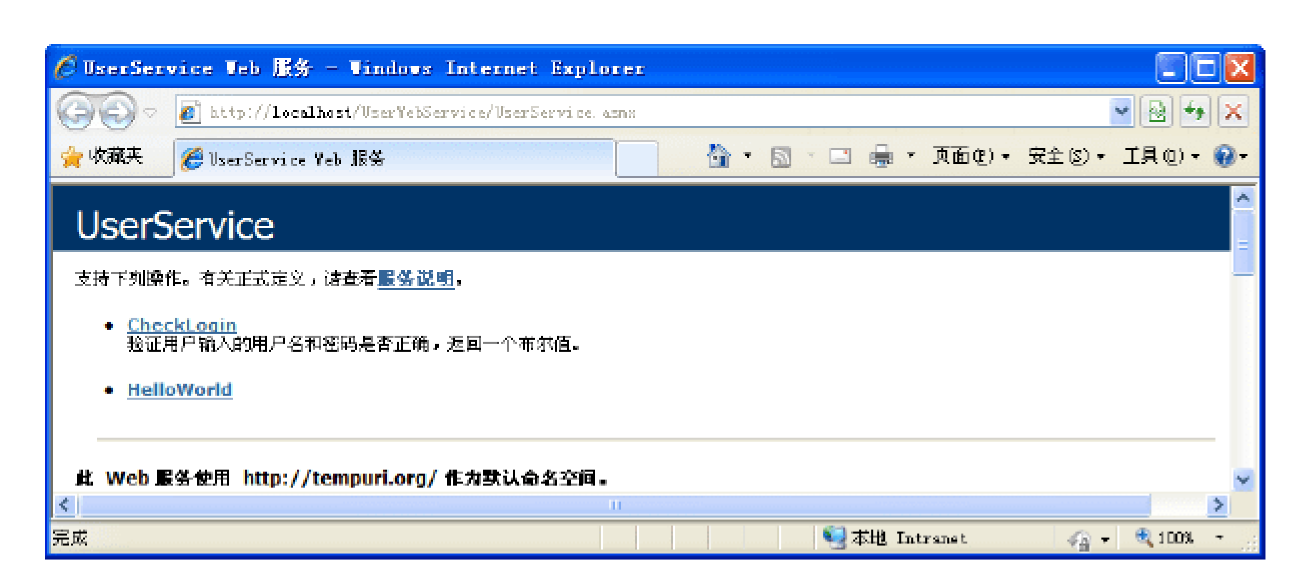


图 2-17 测试 Web 服务

从图 2-17 中可以看到 UserService 是 Web 服务的名字，接下来是 Web 服务可调用的两个方法。这里单击 CheckLogin 链接测试 CheckLogin()方法，如图 2-18 所示。

在文本框内输入字符串，单击【调用】按钮会出现如图 2-19 所示的结果。这样，用 Visual Studio 2010 创建的 Web 服务就完成了。

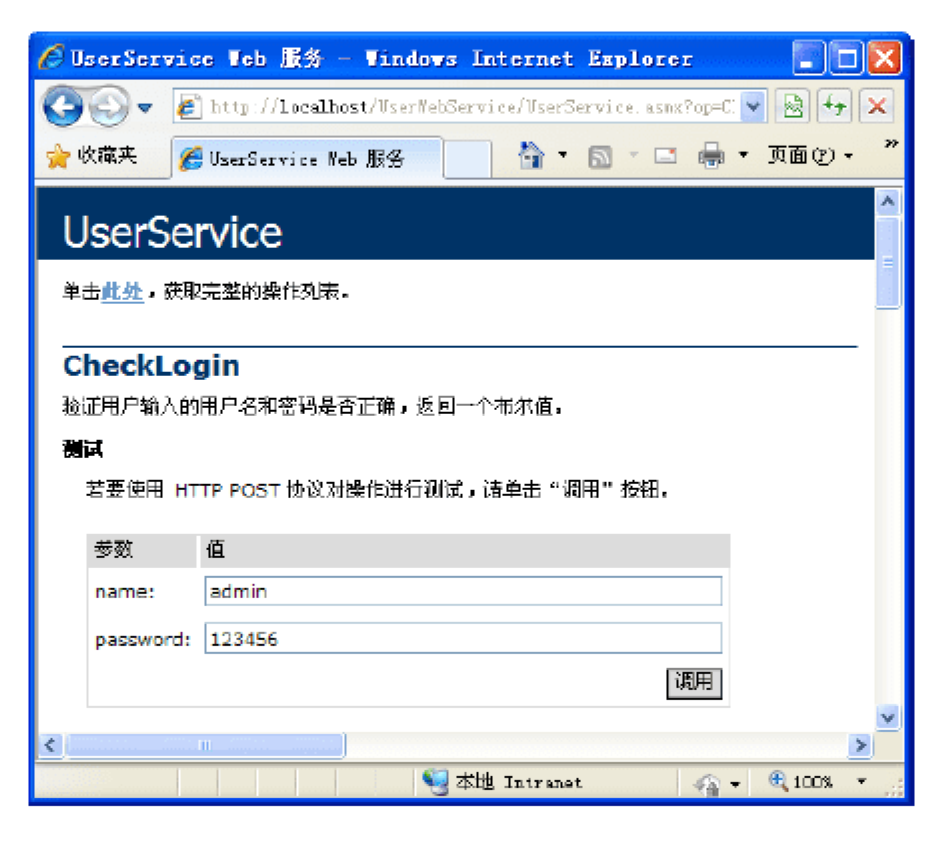


图 2-18　测试 CheckLogin()方法

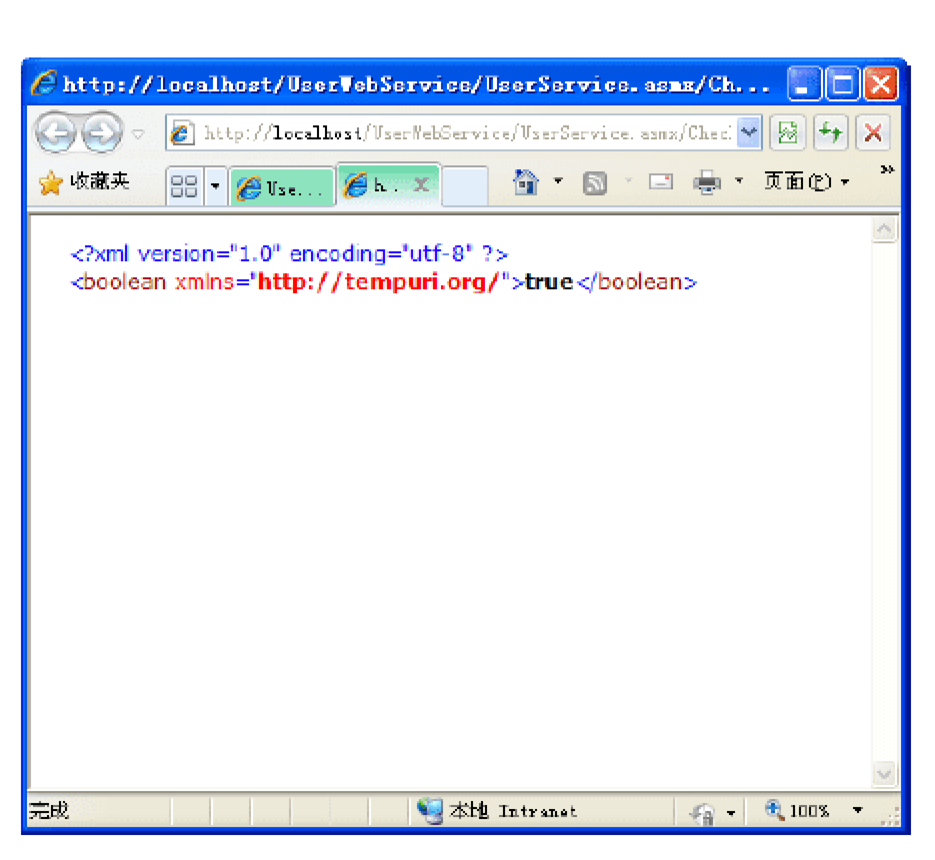


图 2-19　调用结果

2.3.5　实例分析

**源码解析：**

在 Visual Studio 2010 中创建 Web 服务时，在确定其位置时要注意，需要选择 HTTP，并将 Web 服务放在 IIS 虚拟目录下。

然后，Visual Studio 2010 自动生成了创建 Web 服务所需的类以及默认的 Web 服务方法。我们只需按照格式编写代码就可以了，省去了生成以及编译的麻烦。

2.4　使用 ASP.NET 测试 Web 服务

Web 服务的使用非常灵活和自由，在前面我们使用控制台应用程序调用了一个手动生成的 Web 服务，步骤比较繁琐，而且出现错误很难排除。这一节，我们看看利用 Visual Studio 2010 如何在 ASP.NET 中测试 Web 服务。

**视频教学：**光盘/videos/02/ASP.NET 测试 Web 服务.avi　　长度：14 分钟

2.4.1　基础知识——添加服务引用与 Web 引用的区别

由于.NET Framework 4 默认不再推荐 Web 服务，而是通过 WCF 来实现 Web 服务的功能。而.NET Framework 3.5 两者都支持，因此在添加时存在一些差异。

在创建基于.NET Framework 4 的项目或者网站时，右击项目或者网站名称，弹出的快捷菜单中只会出现【添加服务引用】命令，如图 2-20 所示。

选择该项将打开【添加服务引用】对话框，在这里的【地址】下拉列表框中可以输入 WCF 的地址，再单击【前往】按钮。如图 2-21 所示为添加一个 WCF 服务引用时的对话框效果。

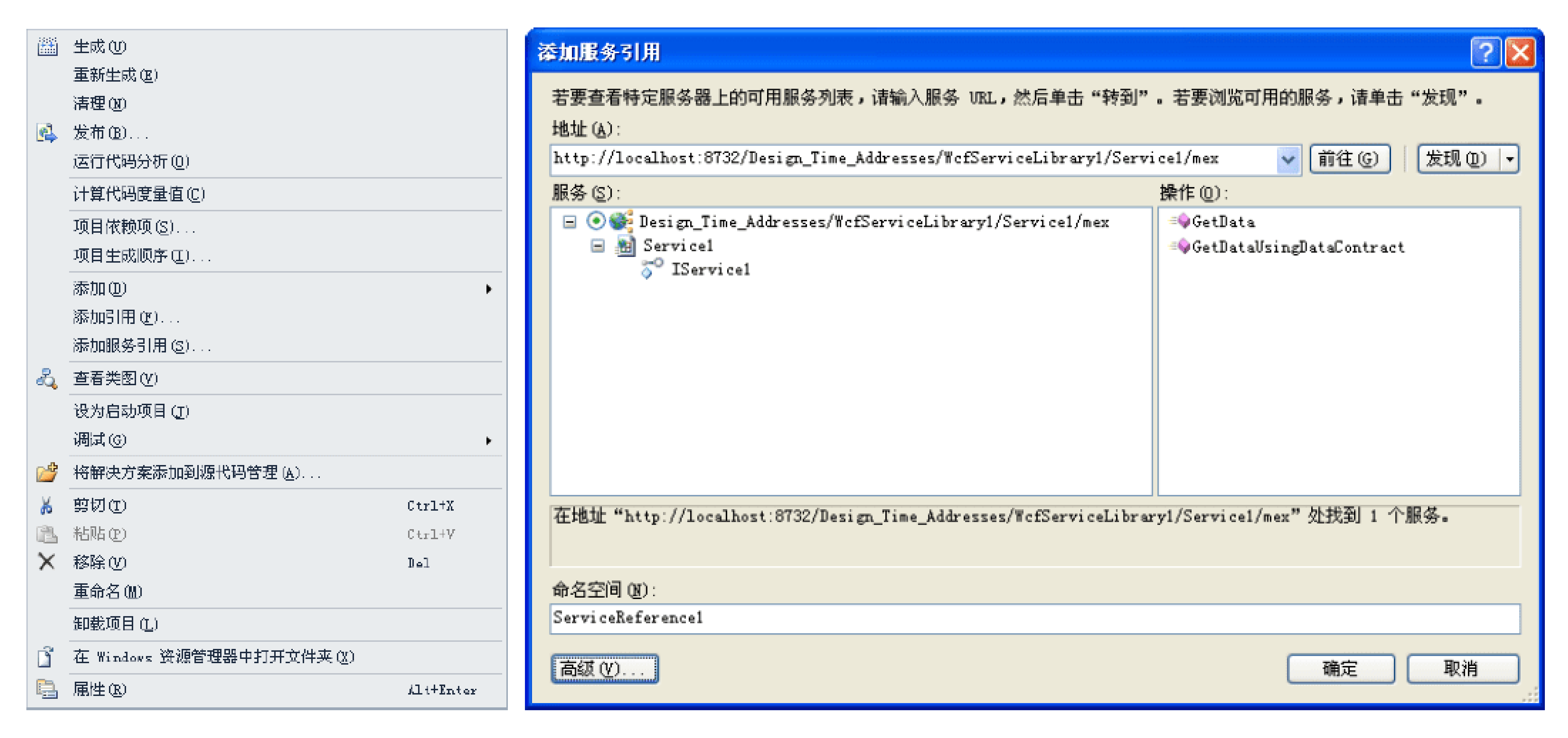


图 2-20　快捷菜单　　　　图 2-21　【添加服务引用】对话框

虽然，也可以通过 Web 服务的 URL 来添加服务引用。但是要注意，此方法不能保证 Web 服务的正常使用，因此不推荐使用。

不过，打开【添加 Web 引用】对话框并不复杂。在图 2-21 所示的对话框中单击【高级】按钮，然后从弹出的【服务引用设置】对话框中单击【添加 Web 引用】按钮即可，如图 2-22 所示。

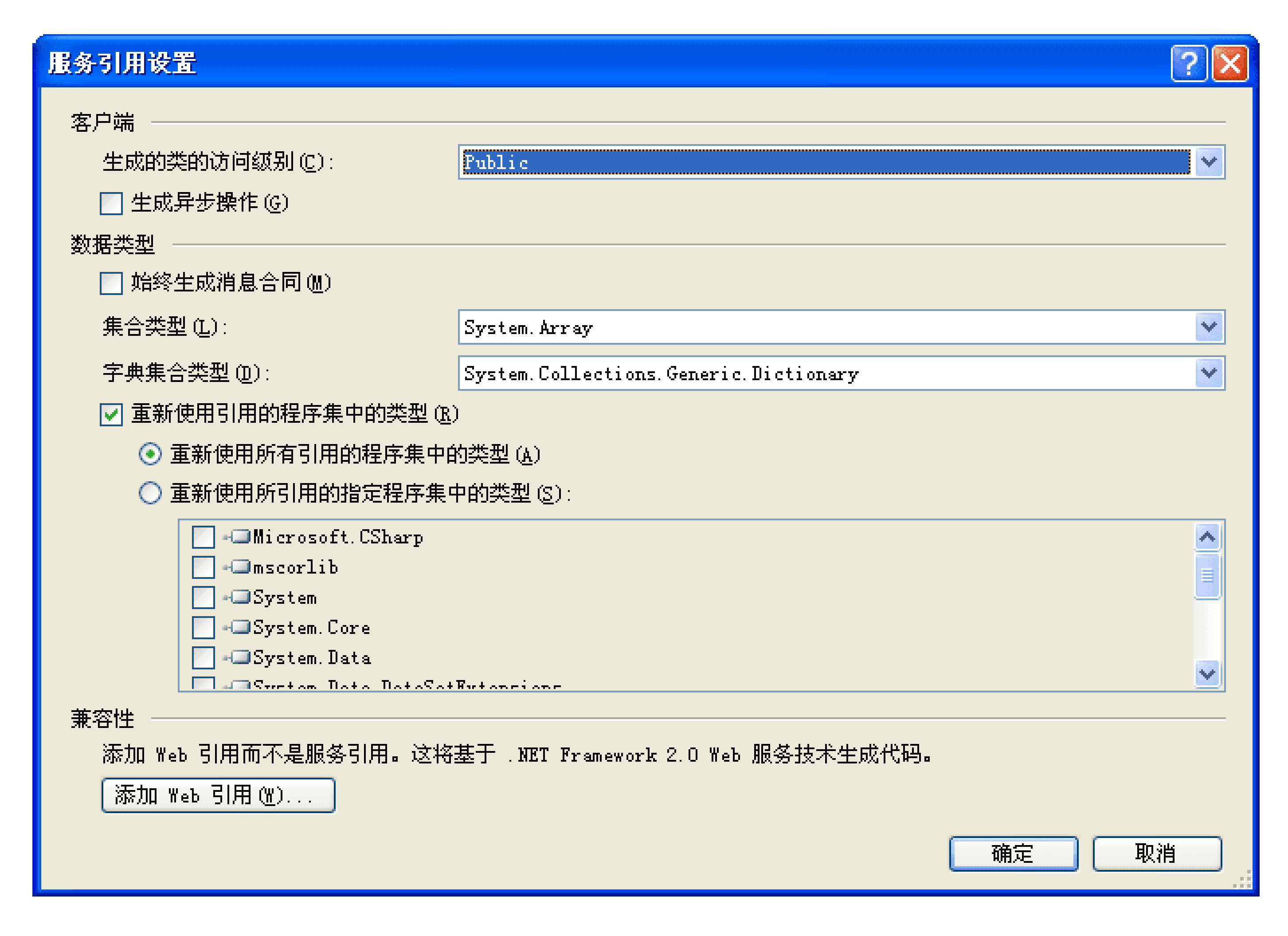


图 2-22　【服务引用设置】对话框

下面我们来了解一下添加服务引用与添加 Web 引用的区别。

(1) 添加服务引用使用的是 WCF 服务，而添加 Web 引用使用的是 Web 服务。

(2) Visual Studio 2010 在升级以后为了支持.NET Framework 3.0 或 3.5 版本上的 WCF Service Library，增加了添加服务引用功能。而对于 Web 服务从.NET Framework 1.0 开始就支持。

(3) 同时存在添加服务引用与添加 Web 引用两者情况的项目类型是 Web 服务程序，包括 Web Service 项目。普通的控制台和窗体等类型是没有添加 Web 引用的。

(4) 使用添加 Web 引用后将由 wsdl.exe 生成客户端代理。而使用添加服务引用后生成客户端代理的命令是 svcutil.exe。

(5) 添加 Web 引用生成的代理可以被.NET Framework 1.0 或者.NET Framework 2.0 的客户端调用。而添加服务引用生成的代理只能被.NET Framework 3.0 以上的客户端调用，而且添加服务引用后不仅生成代理类，在 web.config 中还会生成相应的标记。

(6) 添加 Web 引用生成的 Reference.cs 文件里包含一个服务代理类，它负责与 Web 服务通信。它继承一个 SOAP 类，使用 SOAP 协议，基于 XML 语言。此外还包含一些 Web 服务类里定义的方法，和与之相关的异步调用方法和事件。遵守.NET Web Service 的主要规则。

(7) 添加服务引用生成的客户端文件 Reference.cs 也会反序列化一个本地代理类，这点和前者相似。不过除了服务类和其相关的一些别的类和契约接口外，还有服务请求和相应的信息。遵循 WCF 服务框架的规则。

技术文档	WCF 与 Web 服务的关系
严格地说，Web 服务是行业标准，它有一套规范体系标准，而且在持续不断的更新完善中，也就是 Web Service 规范，也称作 WS-*规范，既不是框架，也不是技术。 微软的 Web 服务实现称为 ASP.NET Web Service，它使用 Soap 来实现分布式环境里应用程序之间的数据交互，用 WSDL 来实现服务接口相关的描述。 WCF(Windows Communication Foundation)是一个分布式应用的开发框架，属于特定的技术，或者平台。既不是标准，也不是规范。WCF 在一定程度上就是 ASP.NET Web Service，因为它支持 Web Service 的行业标准和核心协议。因此 ASP.NET Web Service 能做的事情，WCF 几乎都能胜任，跨平台和语言更不是问题。 但是 WCF 作为微软主推的一个通信平台，它的目标不仅仅是支持和集成 Web Service，因为它还兼容和具备了微软早期很多分布式技术的特性。在本书的第 13 章将详细讨论 WCF。	

2.4.2 实例描述

至此，我们使用 Visual Studio 2010 已经创建了一个 Web 服务，而且也了解了如何在 ASP.NET 中添加对 Web 服务的引用。但还没有涉及如何调用 Web 服务的内容。

那么这一节，我们将基于 2.3 节的 Web 服务创建一个 ASP.NET 项目，并添加 Web 引用实现用户登录与验证的功能。

2.4.3 实例应用

【例 2-4】 使用 ASP.NET 测试 Web 服务

(1) 在 Visual Studio 2010 中打开 2.3 节创建的解决方案，再添加一个 WebApplication 项目。

(2) 右击项目名称执行【添加 Web 引用】命令，在弹出的【添加 Web 引用】对话框中单击【此解决方案中的 Web 服务】链接，如图 2-23 所示。

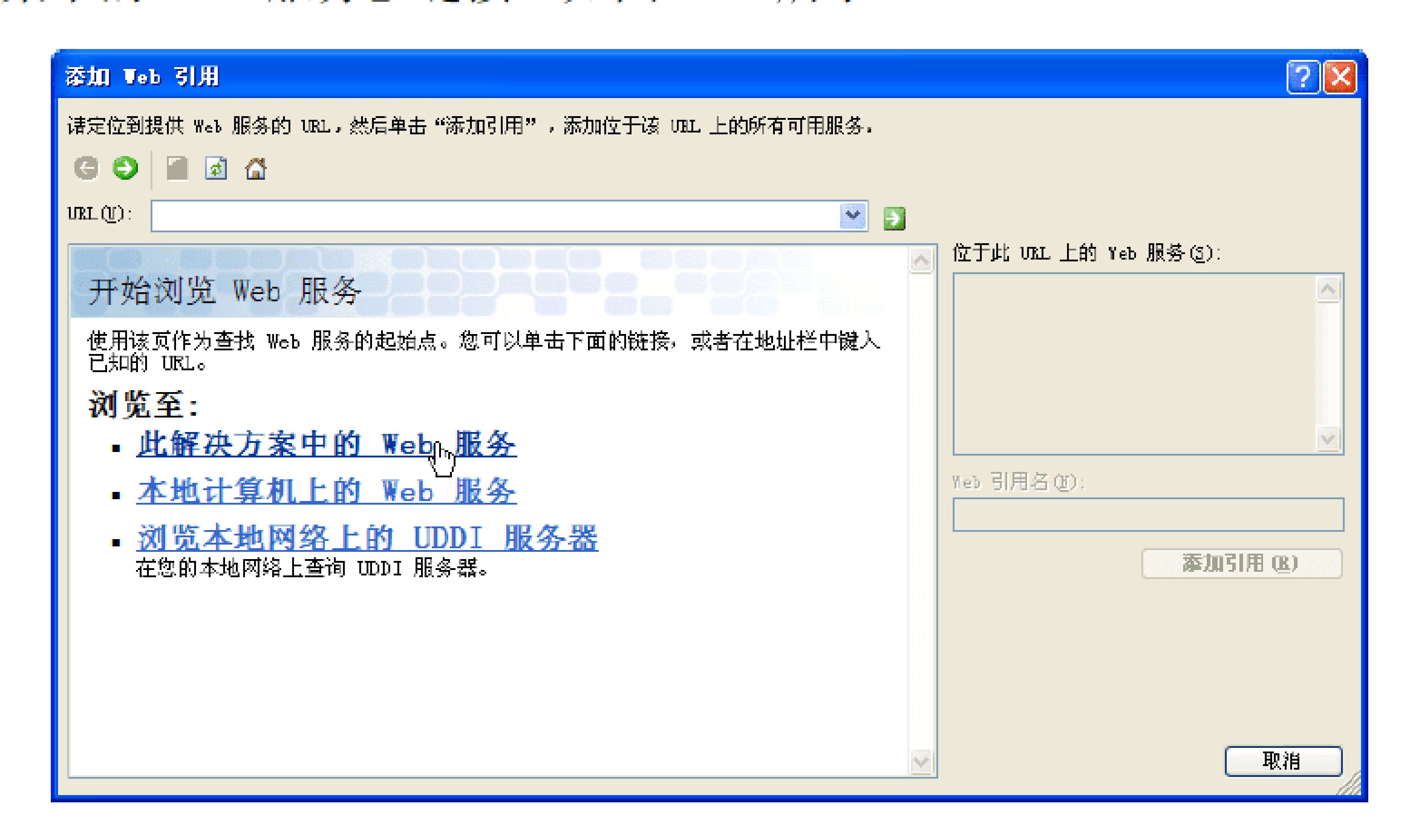


图 2-23 【添加 Web 引用】对话框

(3) 此时 Visual Studio 2010 会自动寻找在当前解决方案下所有项目中可用的 Web 服务并罗列出来。在这里我们会看到有两个结果：UserService 和 Service，如图 2-24 所示。

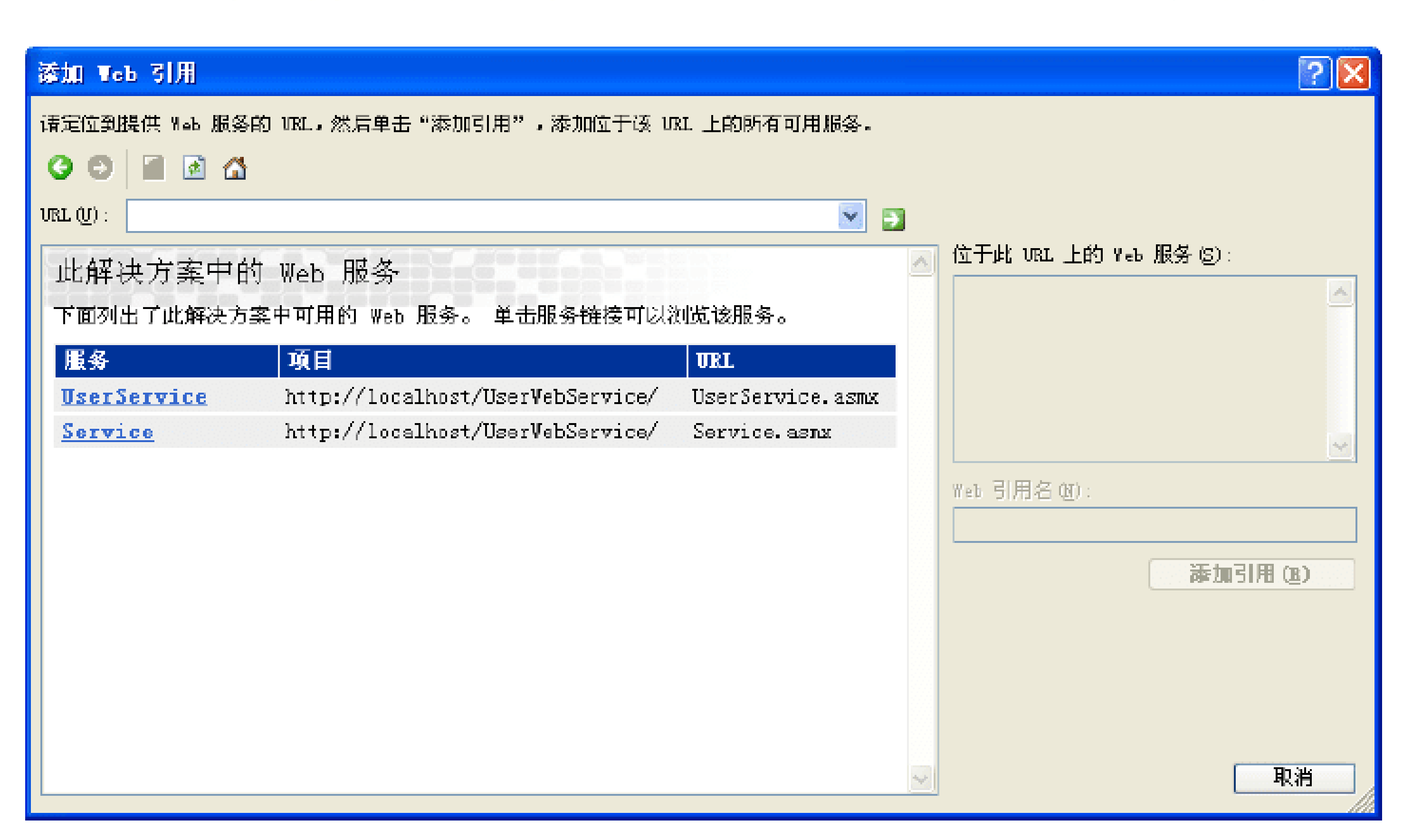


图 2-24 寻找 Web 服务

(4) 单击 UserService 链接进入该 Web 服务的添加引用页面，如图 2-25 所示。

(5) 可以看到，在【Web 引用名】文本框里默认值为“localhost”，它表示我们使用 Web 服务时需要先引用一个命名空间。这里不用修改，单击【添加引用】按钮，Web 服务 UserService 就添加到网站里了。

技巧

我们也可以在【添加 Web 引用】对话框的 URL 中直接输入 Web 服务的地址 http://localhost/UserWebService/UserService.asmx 来转到添加引用页面。

(6) 对 Web 服务的 Web 引用添加完成之后，会看到【解决方案资源管理器】里多出的 Web References 文件夹以及文件，如图 2-26 所示。



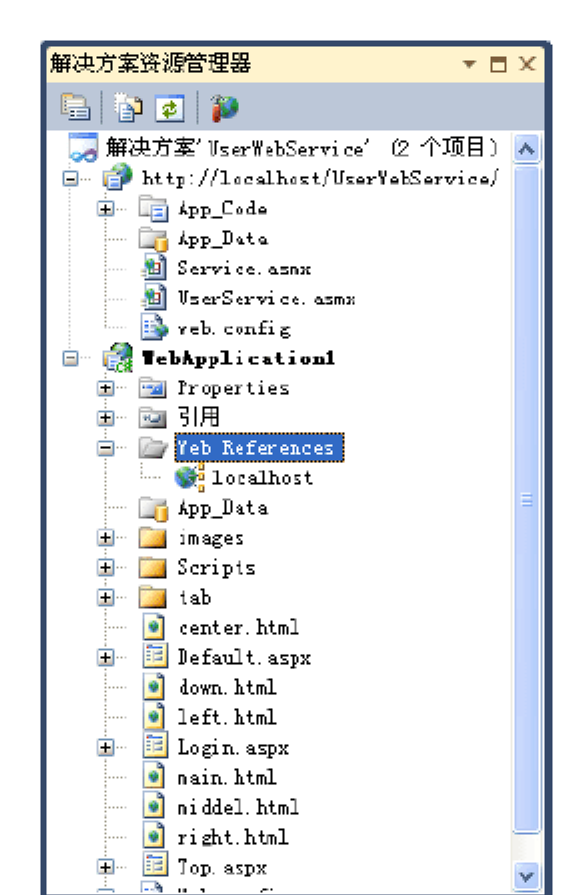


图 2-25　输入 URL 后的添加 Web 引用　　　图 2-26　添加引用后的文件夹和文件

(7) 这样，Web 服务就引用到网站里了。下面通过一个简单的登录来测试 Web 服务是否可用。新建一个名为 Login.aspx 的登录页面，Default.aspx 页面用于显示登录后显示的首页。

(8) Login.aspx 是进入首页之前的登录页面，在这里显示了用户名和密码输入框，以及一个“登录”按钮。页面前台代码如下：

```
<table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td width="16%" height="25"><div align="right"><span class="STYLE1">用户
     </span></div></td>
    <td width="57%" height="25"><div align="center">
     <asp:TextBox ID="txtName" runat="server" style="width:105px; height:17px;
     background-color:#292929; border:solid 1px #7dbad7; font-size:12px;
     color:#6cd0ff"></asp:TextBox>
     </div></td>
     <td width="27%" height="25"> </td>
    </tr>
    <tr>
      <td height="25"><div align="right"><span class="STYLE1">密码
      </span></div></td>
       <td height="25"><div align="center">
            <asp:TextBox ID="txtPwd" runat="server"    style="width:105px;
             height:17px; background-color:#292929; border:solid 1px #7dbad7;
             font-size:12px; color:#6cd0ff" TextMode="Password"></asp:TextBox>

       </div></td>
       <td height="25"><div align="left"> <asp:ImageButton  ID="ImageButton1"
             runat="server" ImageUrl="images/dl.gif" width="49" height="18"
             border="0"
             onclick="ImageButton1_Click" />
          </div></td>
             </tr>
           </table>
```

(9) 进入“登录”按钮的后台单击事件中，编写代码实现调用 Web 服务中的 CheckLogin()方法来验证是否登录，并给出处理方式。代码如下所示：

```
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    //登录
    string name = txtName.Text.Trim();
    string password = txtPwd.Text.Trim();
    localhost.UserService user = new localhost.UserService();
    if (user.CheckLogin(name, password))
    {
        Session["userName"] = name;
        Response.Redirect("Default.aspx");
    }
    else
    {
        Page.ClientScript.RegisterStartupScript(this.GetType(), "",
"<script>alert('登录失败，请查看用户名或密码是否正确。');</script>");
    }
}
```

在上述代码中，localhost 是 UserService Web 服务的 Web 引用名，而 UserService 是要调用的 Web 服务的名字，在创建对象 user 后，就可以通过它来调用 Web 服务里的方法了。

(10) 在 Default.aspx 页面里添加一个 Literal 标签，用来显示信息。然后在后台添加如下实现代码，显示当前登录的用户名：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["userName"] != null)
    {
        ltUser.Text = Session["userName"].ToString();
    }
    else
    {
        Response.Redirect("Login.aspx");
    }
}
```

(11) 保存对文件的修改，完成实例的制作。

2.4.4 运行结果

运行 Login.aspx 文件打开登录页面，如果用户名和密码输入不正确将会弹出错误提示框，如图 2-27 所示。

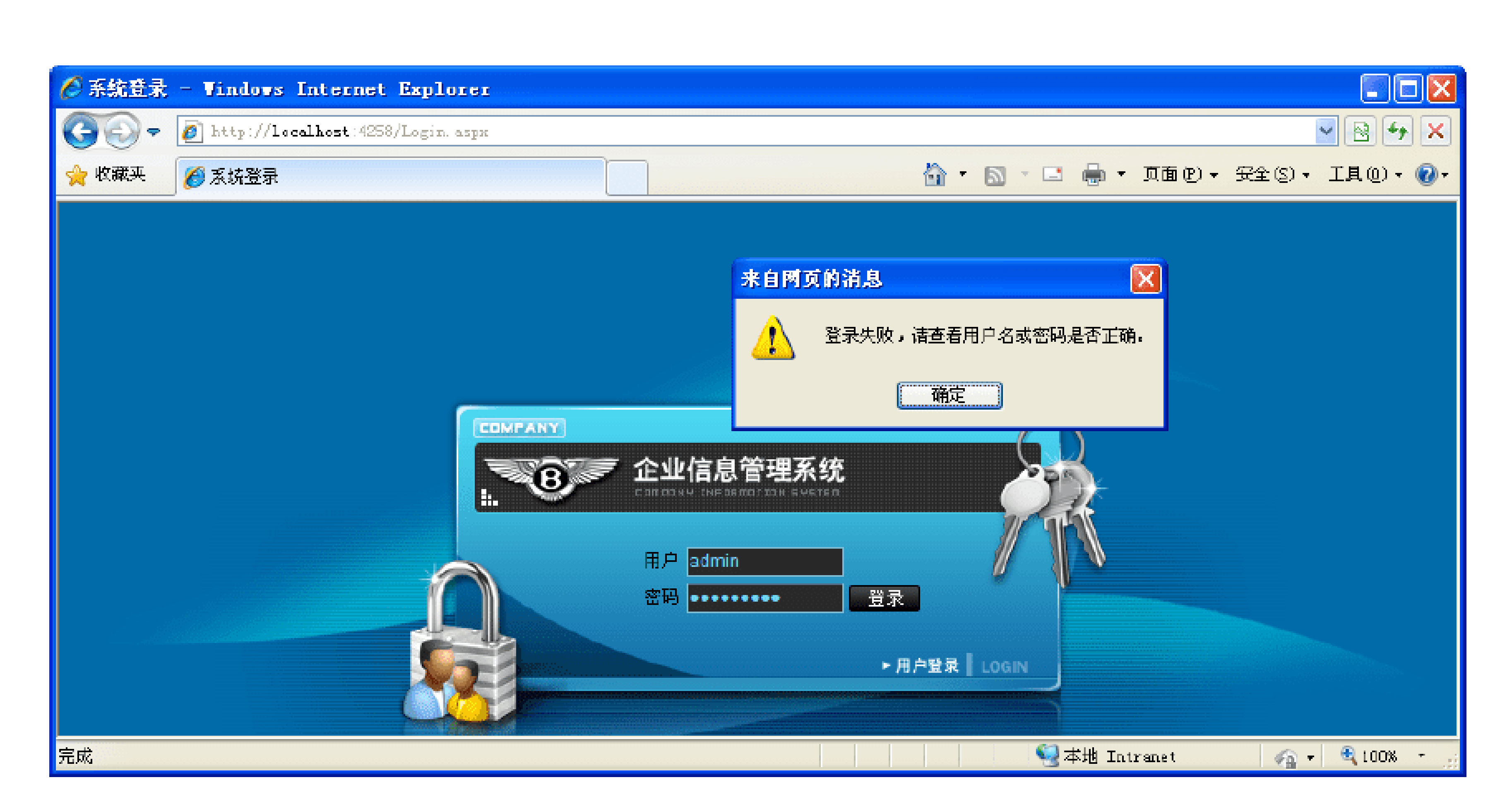


图 2-27　登录页面

如果输入正确，在 Default.aspx 页面会出现 Web 服务返回的字符串，如图 2-28 所示。

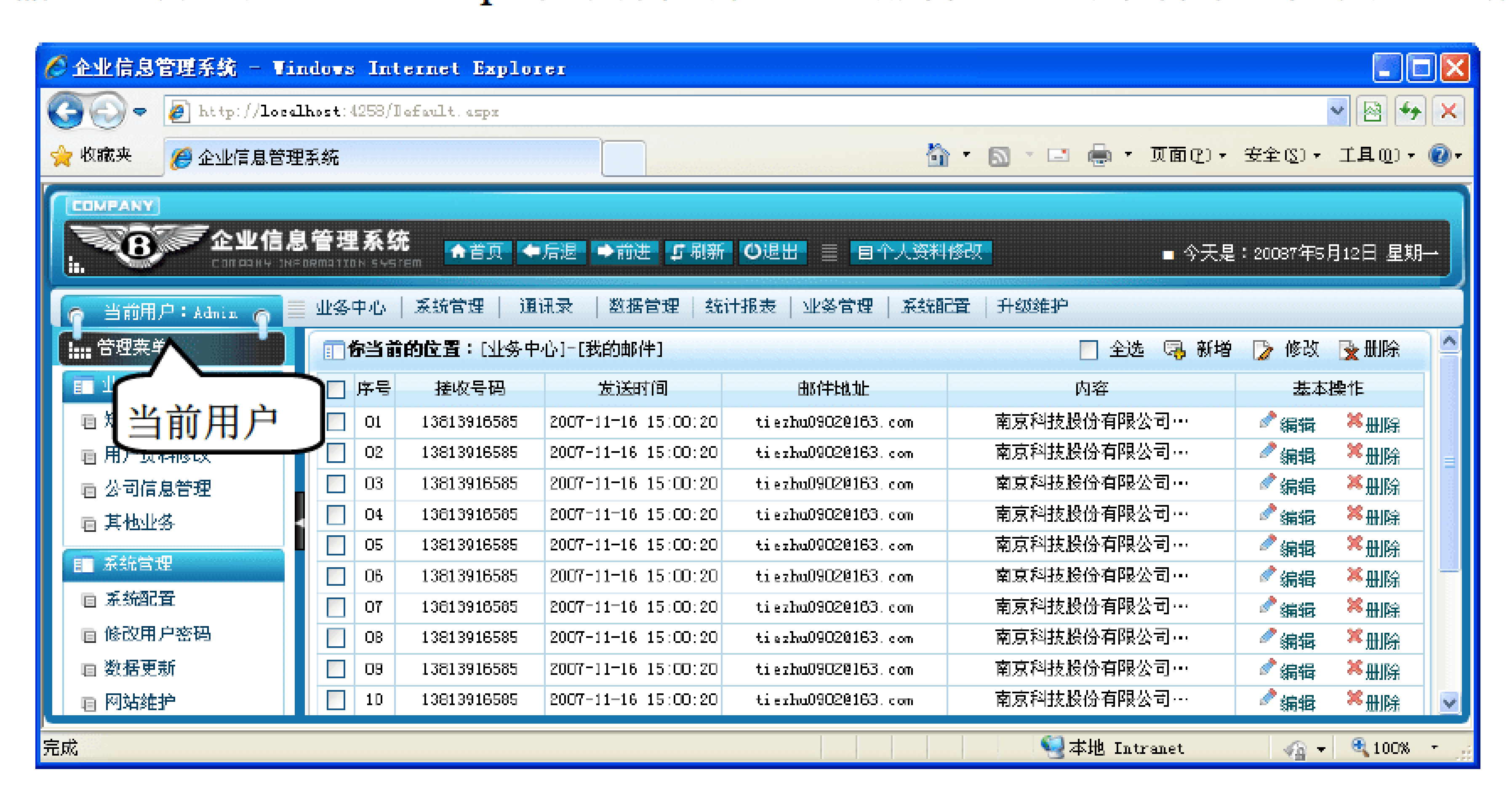


图 2-28　登录后的页面

至此，针对 Web 服务的测试就完成了。

2.4.5　实例分析



源码解析：

本实例主要演示了在 ASP.NET 中调用 Web 服务的过程。这里，我们实现了一个用户登录验证的功能。

但是要注意，添加引用时应该指定一个有意义的名称。然后，使用“引用名称.Web 服务名称”格式来实例化一个 Web 服务类。接下来，便可以像普通类一样调用其中的方法。

另外要创建一个 Web 服务方法，必须使用“[WebMethod]”属性声明。

2.5 创建万年历 Web 服务

使用 WebService 属性，在创建 Web 服务时可以为该服务指定所属的 XML 命名空间。而且还可以使用一个字符串来描述 Web 服务的作用和功能等。

下面我们就来详细了解 WebService 属性的用法。

**视频教学：**光盘/videos/02/创建万年历 Web 服务.avi　　长度：10 分钟

2.5.1 基础知识——WebService 属性

WebService 属性只能用在 Web 服务类中，用于向 Web 服务类添加有关的附加信息。该属性由 System.Web.Service.WebServiceAttribute 类实现，包含的选项有 NameSpace、Name 和 Description，下面分别介绍这些选项。

1. Description

Description 选项用于向 Web 服务添加描述信息，它将成为 WSDL 文档的一部分。例如，下面的示例使用 Description 选项为一个 Web 服务类添加描述文字。

```
[WebService(Description = "<strong>天气预报 Web 服务：包含 2400 个以上中国城市和 100
    个以上国外城市天气预报数据，数据每 2.5 小时左右自动更新一次，准确可靠。</strong>")]
public class WeatherWebServices : System.Web.Services.WebService
```

如上述代码所示，在 WebService 属性中添加了一个 Description 选项，该选项的值为粗体显示的字符串。运行后，这段描述信息将出现在 Web 服务的下方，如图 2-29 所示。

为 Web 服务添加 Description 选项后，ASP.NET 创建 WSDL 文件时，也将在 WSDL 的服务描述文件中的<documentation>标记中显示该描述信息。此时，在图 2-29 所示的页面中单击“服务说明”链接便可以看到，如图 2-30 所示。

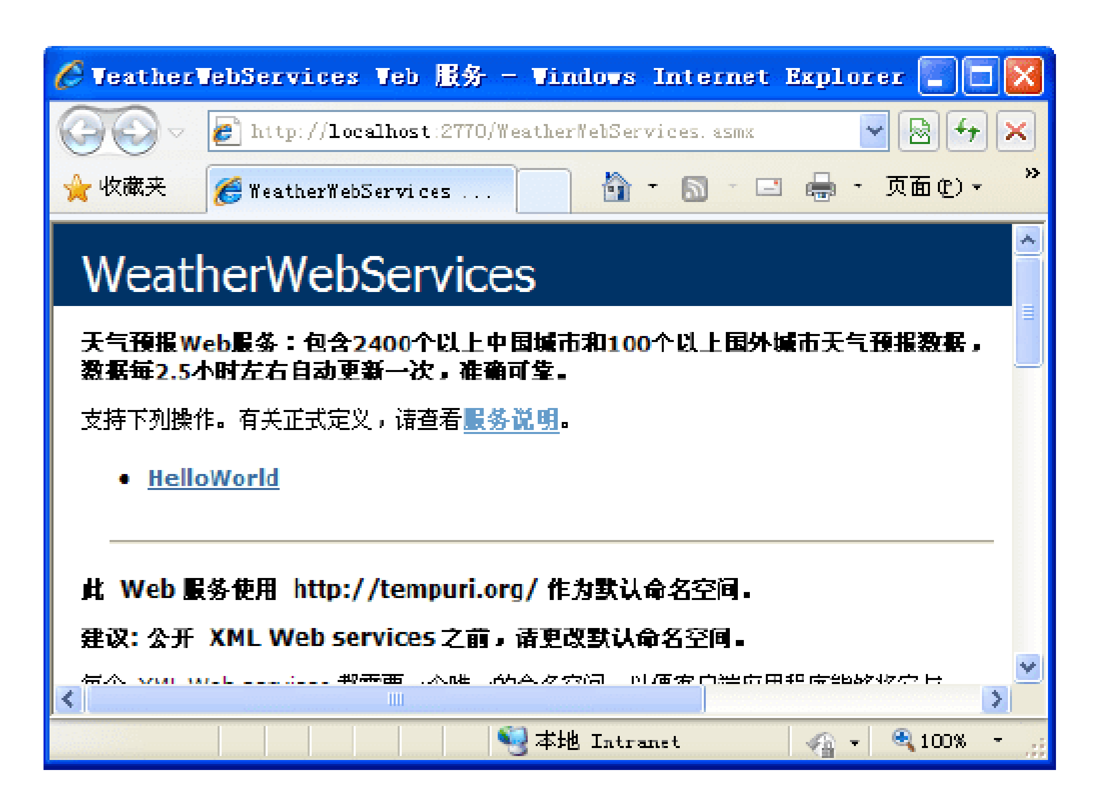


图 2-29 查看 Description 选项

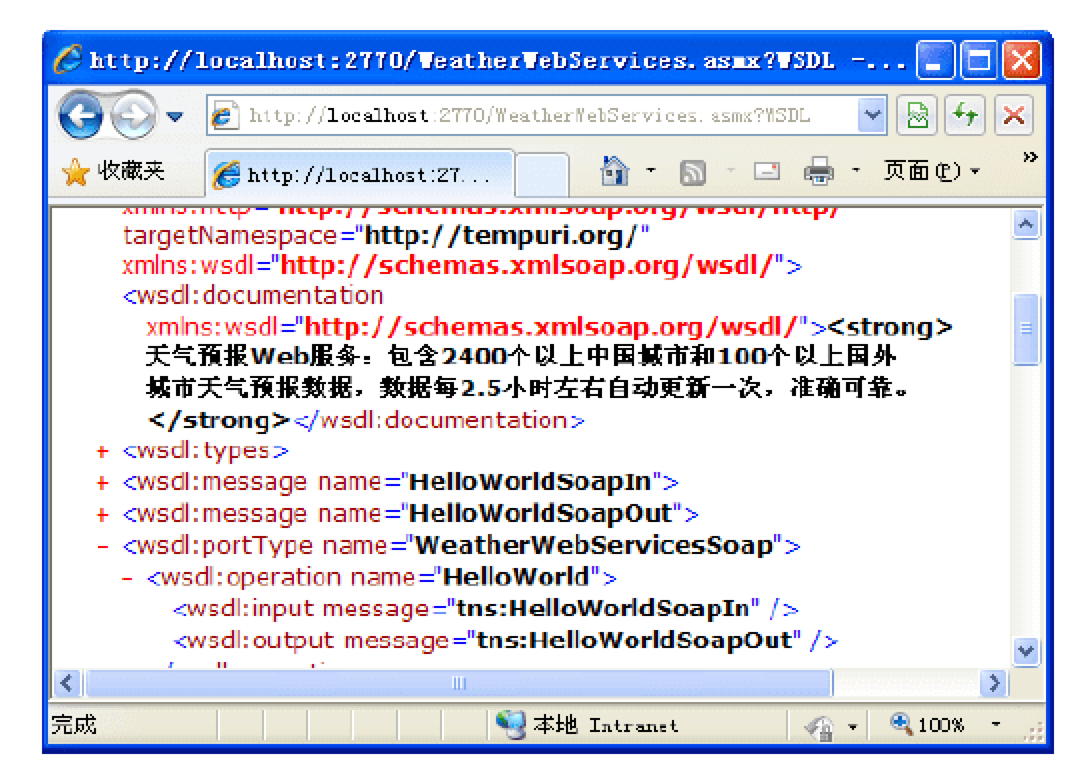


图 2-30 查看修改 Description 选项后的 WSDL

2．Name

默认时，Web 服务的名称与类名相同。Name 选项用于为 Web 服务设置一个与类不相同的名称。如下所示为一个修改后的 Web 服务：

```
[WebService(Name="天气预报的 Web 服务",Description = "<strong>天气预报 Web 服务：包
    含 2400 个以上中国城市和 100 个以上国外城市天气预报数据，数据每 2.5 小时左右自动更新一次，
    准确可靠。</strong>")]
public class WeatherWebServices : System.Web.Services.WebService
```

这里使用 Web 服务的 Name 选项，为 Web 服务指定了一个与类名不相同的名称，测试页面如图 2-31 所示。在 Web 服务的描述文件 WSDL 中也将随之改变，此时效果如图 2-32 所示。



图 2-31　测试 Name 选项

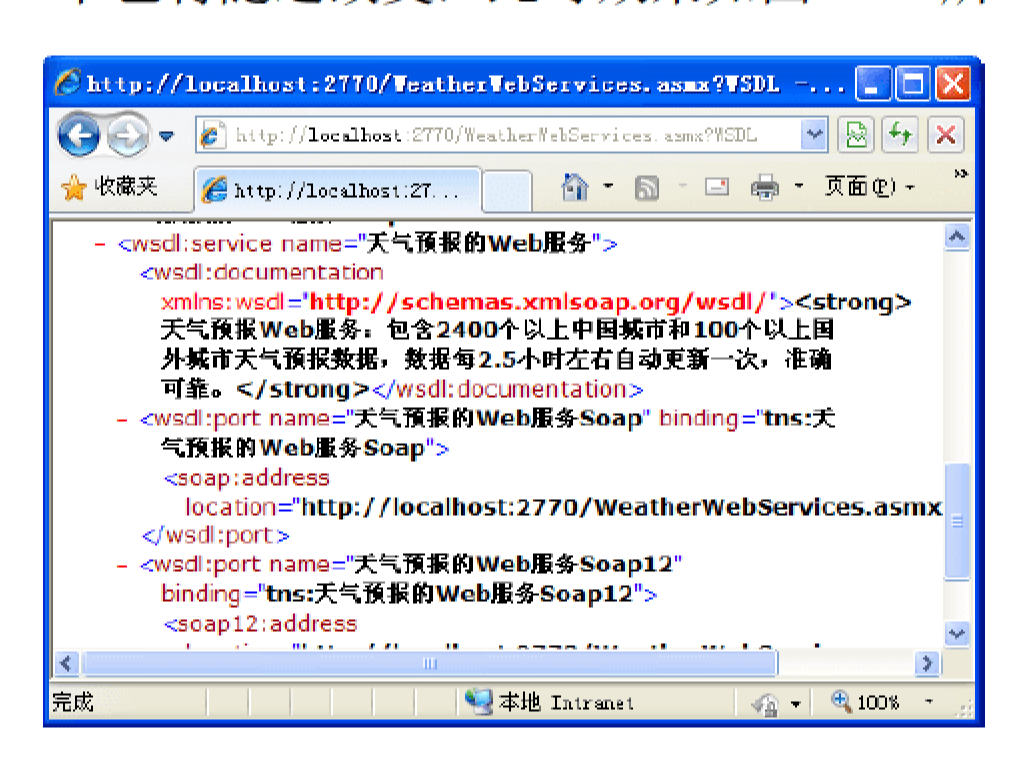


图 2-32　查看修改 Name 选项后的 WSDL

3．NameSpace

默认时，我们看到所有生成的测试页都有这样一段文字“此 Web 服务使用 http://tempuri.org/ 作为它的默认命名空间。”同时，在测试页中还说明了如何修改命名空间。可以通过 WebService 属性的 NameSpace 选项来完成这一工作，如下所示：

```
[WebService(Namespace="www.iWebServices.com",Name="天气预报的 Web 服务",
    Description = "<strong>天气预报 Web 服务：包含 2400 个以上中国城市和 100 个以上国外
    城市天气预报数据，数据每 2.5 小时左右自动更新一次，准确可靠。</strong>")]
public class WeatherWebServices : System.Web.Services.WebService
```

除了 Description 和 Name 选项值的修改可以在 WSDL 文件中自动更新外，NameSpace 选项也不例外，如图 2-33 所示为添加命名空间后的 WSDL 文件效果。

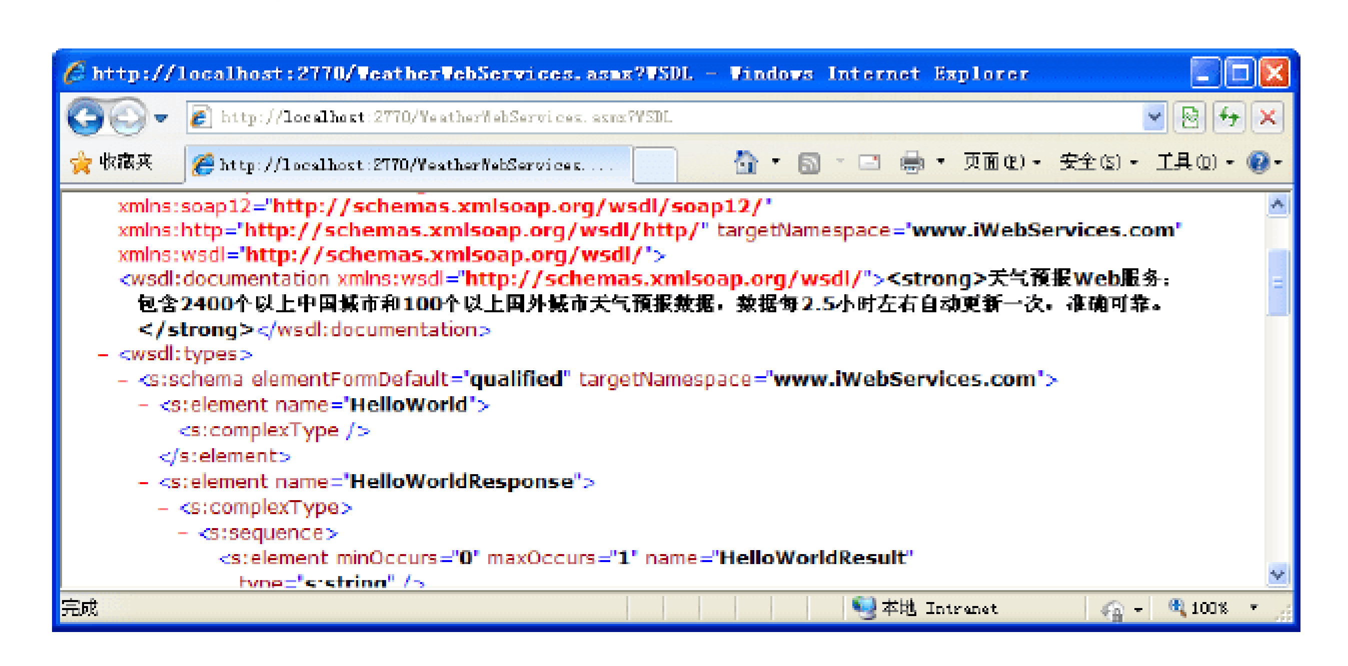


图 2-33　查看修改 NameSpace 选项后的 WSDL



WebService 属性不是创建 Web 服务必需的，该属性及其属性项被添加到 Web 服务的标题中，标题与编译的代码一起保存在程序集中。

2.5.2 实例描述

作为一名开发者，应该尽量从用户角度出发，才能开发出用户体验好、操作快捷的应用程序。

对于 Web 服务的开发也是如此。一个准确的 Web 服务名称，再配上简短的描述文字，可以让 Web 服务的调用者更好地了解和使用它。

下面就以做 OA 系统时用到的万年历 Web 服务为例，来了解如何为 Web 服务添加描述。

2.5.3 实例应用

【例 2-5】 创建万年历 Web 服务

(1) 打开 2.5.1 节 Web 服务 WeatherWebServices.asmx 所在的解决方案。

(2) 通过【添加新项】对话框增加一个名为“wannianliWS.asmx”的 Web 服务。

(3) 打开源代码文件 wannianliWS.asmx.cs，可以看到系统自动生成如下代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
namespace WebService1
{
    /// <summary>
    /// wannianliWS1 的摘要说明
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // 若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
    // [System.Web.Script.Services.ScriptService]
    public class wannianliWS1 : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

上述代码中，加粗部分的 Namespace 属性表示当前 Web 服务所在的命名空间。如果不修

改，那么 ASP.NET 将默认使用“http://tempuri.org/”，但此名称只能在开发过程中使用，实际应用时必须进行修改。

(4) 然后添加 Name 和 Description 属性，代码如下：

```
[WebService(Namespace = "http://www.myWebService.com/",Name="万年历的 Web 服务",Description="万年历的 Web 服务：提供最精准的万年历服务，节气黄历一应俱全，每日忌宜，详细准确")]
```

(5) 保存对文件的修改，完成实例的制作。

2.5.4　运行结果

运行后效果如图 2-34 所示。为 Web 服务提供的名称显示在页面最上方，为 Web 服务添加的描述文字则显示在 Web 服务名称的下面。

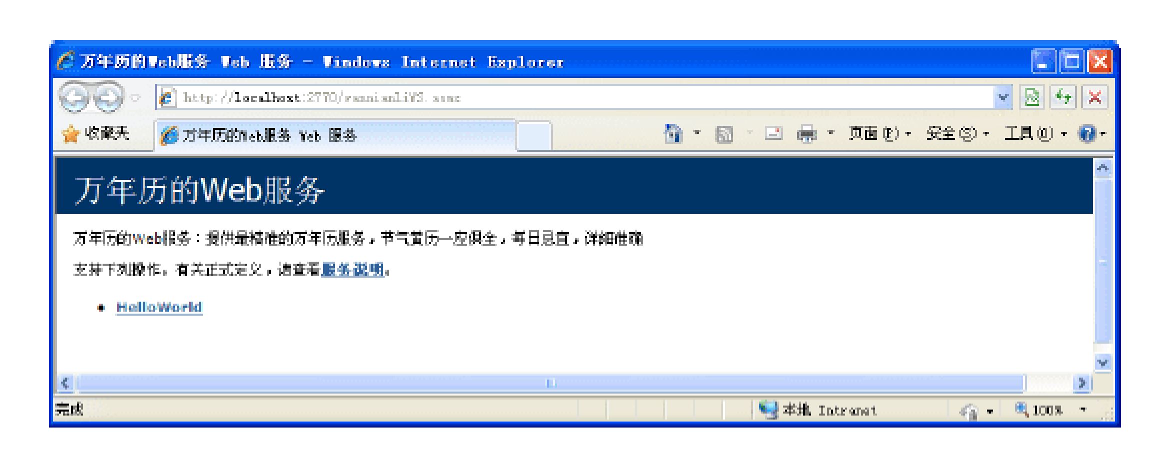


图 2-34　查看万年历 Web 服务

接下来查看 Web 服务的描述文件，单击【服务说明】链接打开的详细页面，如图 2-35 所示。

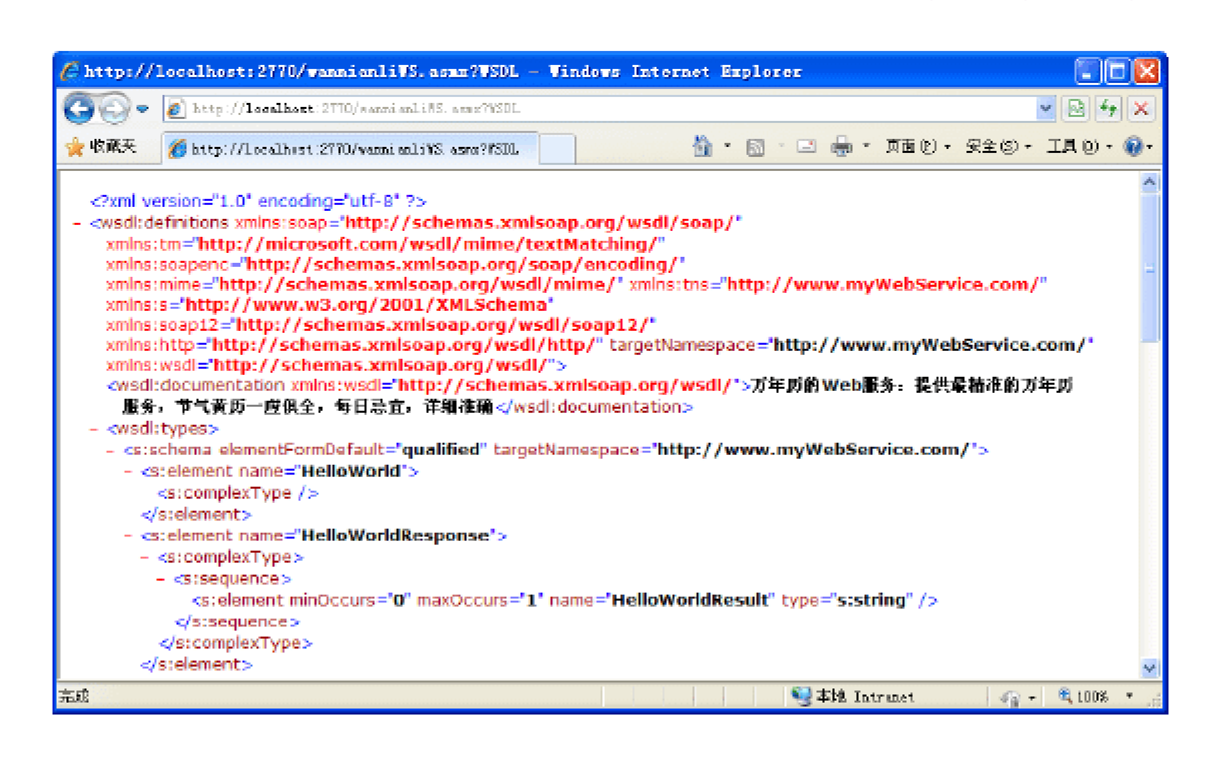


图 2-35　服务说明的内容

2.5.5　实例分析


源码解析：

WebService 属性在实际开发时应用频率非常高，这也与它的 3 个选项分不开。Namespace 唯一指定了 Web 服务所在的命名空间，Name 选项可以指定一个别名，而使用 Description 选项能够为 Web 服务指定一段描述它的功能的信息。

2.6　为 Web 服务方法添加说明

在上一节中，我们学习了如何对 Web 服务类进行修饰。在这个 Web 服务类中会有很多方法，但不是所有的方法都可以通过 Web 进行调用，而只有使用 WebMethod 属性修饰的方法才可以。

另外，要成为 Web 服务方法除了带有 WebMethod 属性外，还必须声明为 public 方法。WebMethod 属性提供了很多的选项来控制 Web 服务方法的行为，本节我们就来了解它们。

 **视频教学：**光盘/videos/02/为 Web 服务方法添加说明实验.avi　　长度：10 分钟

2.6.1　基础知识——WebMethod 属性

WebMethod 属性由 System.Web.Services.WebMethodAttribute 类实现，它包含 6 个选项来说明和更改 Web 方法的行为，分别是 CacheDuration、Description、EnableSession、MessageName、TransactionOption 和 BufferResponse。下面通过一些简单的示例介绍这些选项。

1．CacheDuration

在 Web 服务中实现适当的缓存可以提高可扩展性和性能。实现缓存系统最容易的方法之一是使用 WebMethod 属性的 CacheDuration 选项。

使用 CacheDuration 选项可以设置请求/响应对在缓存中存储的秒数(整数)，它的默认值为 0，表示不缓存响应。对于涉及需要大量处理器资源的查询或者结果不经常发生变化并且开销较大的其他查询的 Web 来说，这种缓存机制是非常理想的。例如，在一个商城系统中，获得每个用户订单信息的 Web 方法就是这种功能的例子。对于这样的系统来说，我们可以将 Web 方法的 CacheDuration 属性设置为 1 分钟或者更长的时间，从而减少到数据库的往返过程。

如下所示代码，演示了如何为一个 Web 方法添加 CacheDuration 选项。

```
[WebMethod(CacheDuration=60)]
public DataSet getOrderDetailsById(string ordId) {
    //这里是具体的实现，此处省略
}
```

2．EnableSession

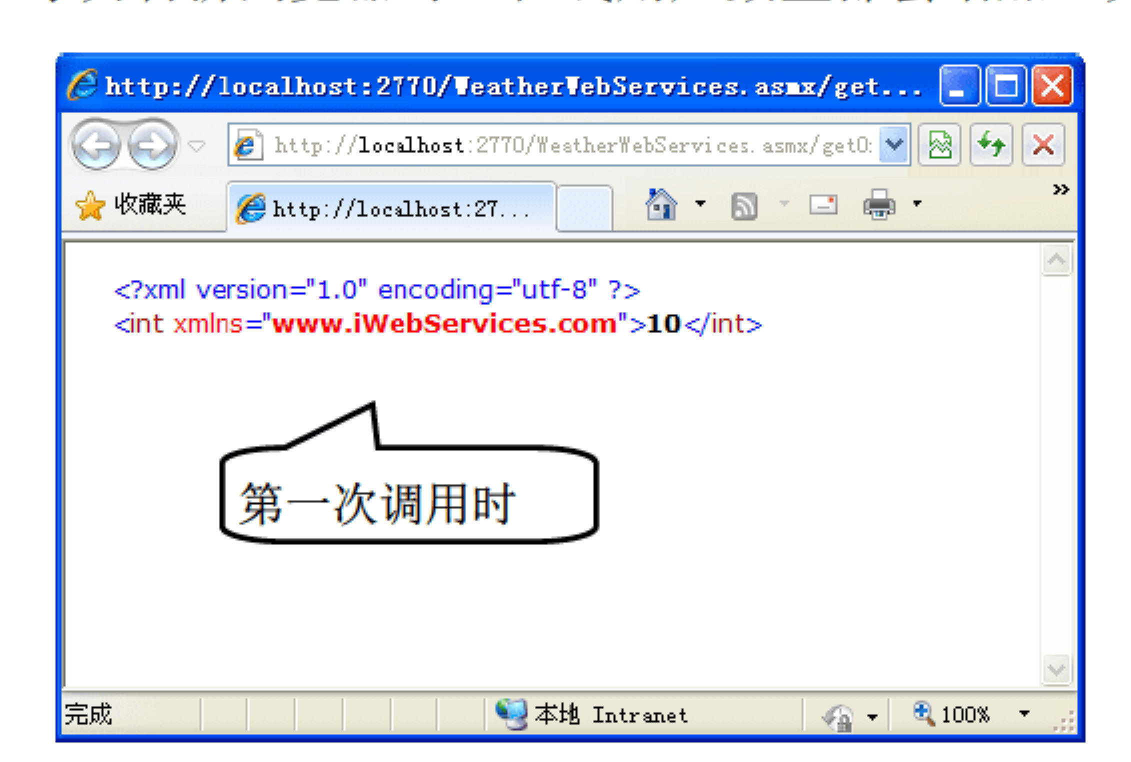
ASP.NET Web 应用程序最大的优点就是可以禁用会话状态。这一优点也适用于 Web 服务。默认时，Web 服务不支持会话状态。大多数 Web 服务被设计成与状态无关的，以便实现 Internet 的可伸缩性，因为处于会话状态时服务器上的每一个客户都会消耗内存。

但是，有时我们可能希望 Web 服务支持会话状态。我们可以对每一个方法启用会话状态。要想对 Web 服务方法启用会话状态，可以使用 WebMethod 属性的 EnableSession 选项。如下代码演示了获取当前在线用户数量的方法。

```
[WebMethod(EnableSession=true)]
public int getOnlineUsers() {
    int count;
    if (Session["users"] == null)
    {
        count = 10;
    }
    else
    {
        count = (int)Session["users"] + 100;
    }
    Session["users"] = count;
    return count;
}
```

这里通过设置 EnableSession 选项的值为 true 来启用会话状态。在 getOnlineUsers()方法中使用 Session 对象来获取当前在线用户的数量。

用户在使用 ASP.NET 测试页从浏览器调用 getOnlineUsers()方法时，将会获得希望的结果。每次刷新浏览器时，在线用户数量都会增加，如图 2-36 所示。



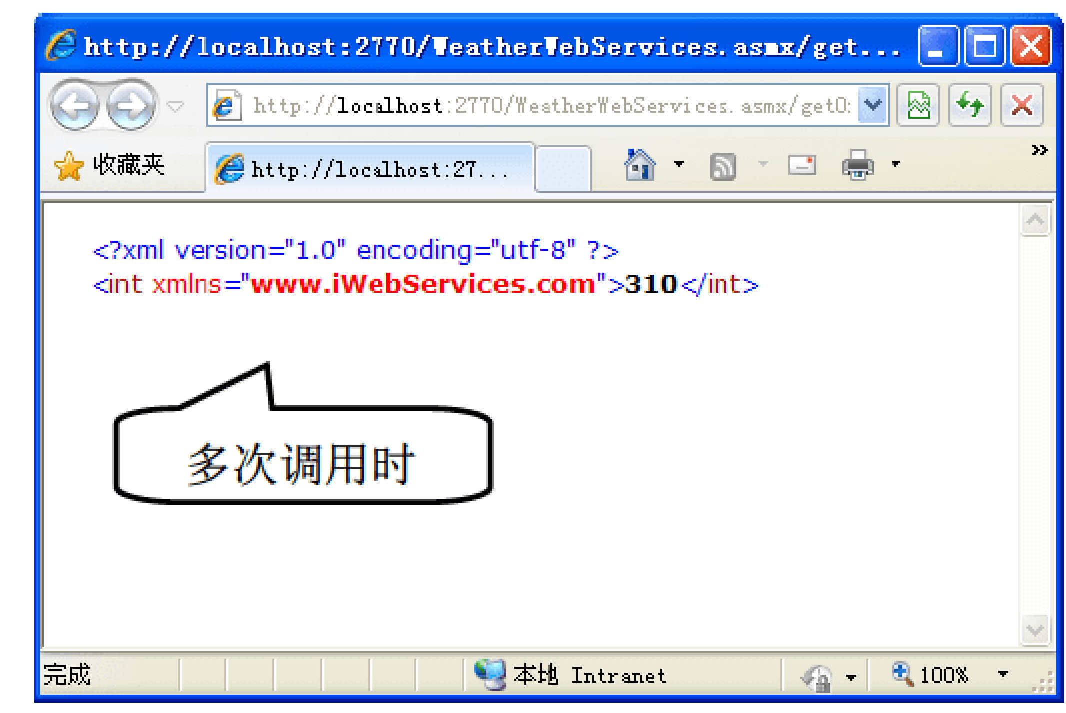


图 2-36　使用 EnableSession 选项的 getOnlineUsers 方法

3．Description 选项和 MessageName 选项

为了避免用户根据 Web 服务方法的名称来猜测它的作用，我们可以对每一个 Web 服务方法提供一个描述。当 Web 服务包含重载的 Web 服务方法时，这样做是特别有必要的。

例如，在下面给出的代码中声明了两个名为 Calculate 的方法，一个接受整型参数，另一个接受单精度浮点型参数。

```
[WebMethod]
public int Calculate(int x,int y)
{
    return x + y;
}
[WebMethod]
public float Calculate(float x, float y)
{
```

```
    return x + y;
}
```

这段程序中，使用重载的方法创建了两个相同名称的方法 Calculate，但是两个方法的参数不同。那么在查看这个页面时会出现运行时异常，如图 2-37 所示。

如图 2-37 所示，异常信息中提示同时使用了消息名称 Calculate。这种情况下，可以为每个重载的方法使用 WebMethod 属性。再使用该属性的 Description 选项为 Web 服务的方法添加描述信息，使用 MessageName 选项更改它的名称。如下所示为修改后的代码：

```
[WebMethod(MessageName="CalculateInt", Description="求两个整数的和")]
public int Calculate(int x,int y)
{
    return x + y;
}
[WebMethod(MessageName = "CalculateFloat", Description = "求两个单精度数的和")]
public float Calculate(float x, float y)
{
    return x + y;
}
```

然后再打开 Web 服务的测试页，即可看到修改后的效果如图 2-38 所示。

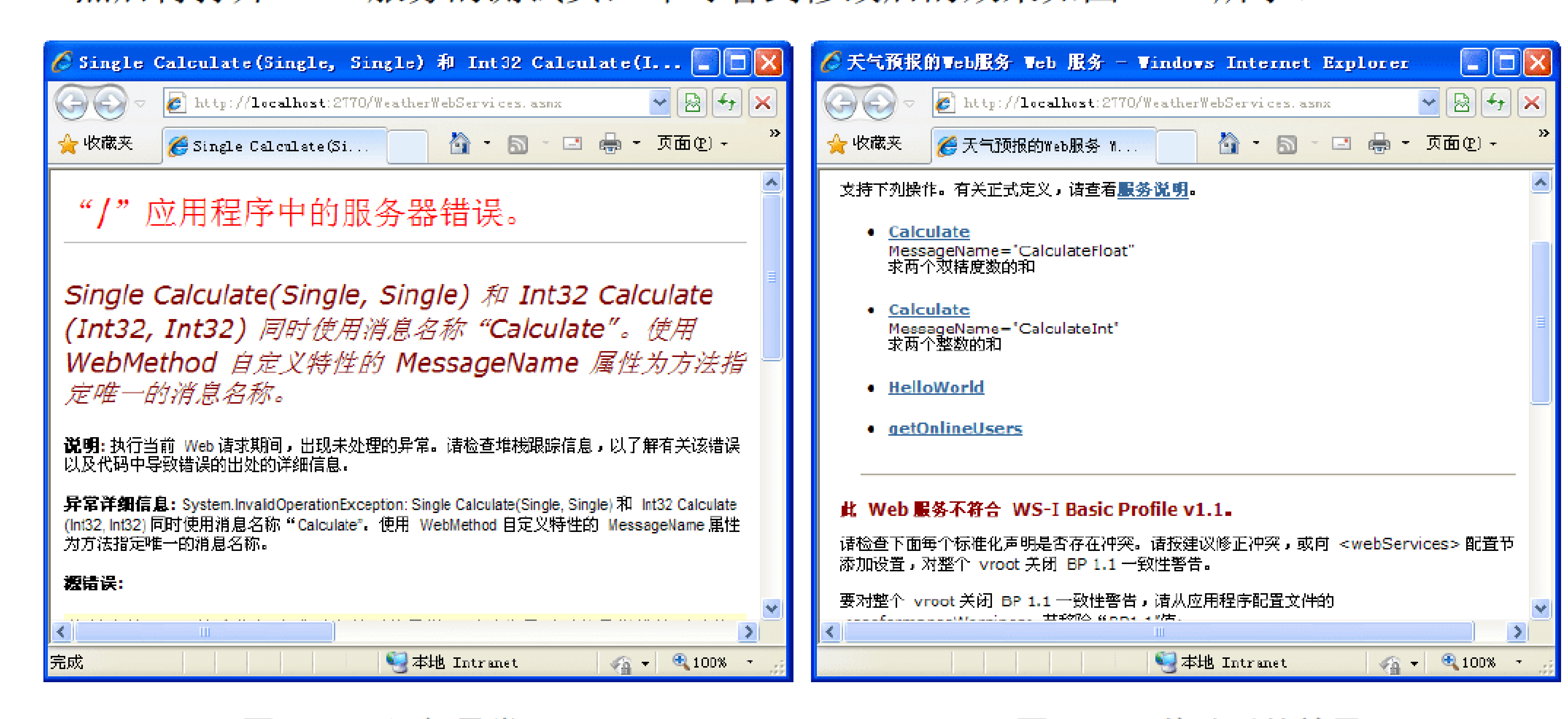


图 2-37　运行异常　　　　图 2-38　修改后的效果

4．BufferResponse 选项

Web 方法的默认行为是在内存缓冲区中存储响应，直到将缓冲区填满或者响应完成为止，这个存储过程被称为序列化。在大多数情况下，这种行为是可行的，因为缓冲可以减少客户的传输数量，从而得到更高的性能。

但是，如果 Web 方法返回大量数据或者用很长时间来运行，那么客户可能想要将 BufferResponse 选项设置为 False，来禁用缓冲。这个设置会将响应立即发送到客户端，但是由于结果集更小，因此这可能会降低性能。

如下代码演示了一个使用 BufferResponse 选项的例子：

```
[WebMethod(BufferResponse = false)]
public DataSet ExecuteSql(string sql)
{
    //执行数据库查询操作返回所有数据集
}
```

5．TransactionOption 选项

在 ASP.NET 中页面支持事务处理功能。只需用一个事务处理属性标识 ASP.NET 页面，该页面中的所有代码就将处于一个事务处理中。

ASP.NET Web 服务支持相同的模式。我们可以把 Web 服务方法标识为支持事务处理功能，然后该方法中的全部代码都将处于一个事务中。

如果要为 Web 方法设置事务处理支持，可以像下面的示例一样使用 WebMethod 的 TransactionOption 选项。

```
using System.EnterpriseServices;
[WebMethod(TransactionOption=TransactionOption.Required)]
 public UpdateUserAccount(float money)
 {
     //此 Web 服务方法将支持事务
 }
```

由于事务处理服务位于 System.EnterpriseServices 命名空间。因此，在使用时用户首先需要在项目中添加对 System.EnterpriseServices.dll 文件的引用。

这里 TransactionOption 选项是一个枚举类型的值，它位于 System.EnterpriseServices. TransactionOption 命名空间，可选值有 5 个：Disabled、NotSupported、Required、RequiresNew 和 Supported。



Web 服务不能参与正在进行的事务处理。它通常会启动一个新的事务处理。Web 服务与客户机不能共享事务处理的内容。

2.6.2　实例描述

无论采用哪种方式，当我们创建好一个 Web 服务类之后，剩下的工作就是添加 Web 服务方法。为了使一些方法能够在客户端调用，需要使用 WebMethod 属性进行声明。

WebMethod 属性提供了很多选项来控制一个 Web 服务方法。当然，它们都是可选的，而且都有默认值。

但是，作为一名开发人员只有深入了解各个选项的含义，才能开发出适合不同情形的 Web 服务方法。

2.6.3　实例应用

【例 2-6】　为 Web 服务方法添加说明

(1) 打开 2.5.2 节 Web 服务 WeatherWebServices.asmx 所在的解决方案。

(2) 通过【添加新项】对话框增加一个名为“myWebService.asmx”的 Web 服务。

(3) 打开 Web 服务 myWebService.asmx.cs 后台代码文件。添加一个不带参数的 getInfo() 方法，实现代码如下所示：

```
[WebMethod(Description = "用于判断单击次数", EnableSession = true, MessageName
   = "getInfoByClick")]
public string getInfo()
{
   int count;
   System.Web.SessionState.HttpSessionState Session;
   Session = System.Web.HttpContext.Current.Session;
   if (Session["clickcount"] == null)
      count = 1;
   else
      count = (int)Session["clickcount"] + 1;
   Session["clickcount"] = count;
   if (count >= 4)
   {
      return "系统忙，请稍候再试";
   }
   else
   {
      return "没有数据";
   }
}
```

(4) 再添加一个同名的 getInfo()方法，这里为它设置一个字符串参数，实现代码如下：

```
[WebMethod(Description = "据名称查询是否存在该用户", MessageName =
"SelUserInfoByName", TransactionOption = TransactionOption.Required)]
public string getInfo(string name)
{
   string conn;
   string sql;
   SqlDataReader reader;
   conn = "Data Source=.;Initial Catalog=Login;User ID=sa;Password=123";
   sql = "select * from LoginInfo where name ='" + name + "'";
   try
   {
      SqlConnection con = new SqlConnection(conn);
      SqlCommand com = new SqlCommand(sql, con);
      con.Open();
      reader = com.ExecuteReader();
   }
   catch (Exception e)
   {
      return e.Message;
   }
```

```
    if (reader.Read())
    {
        return "用户" + name + "存在！";
    }
    else
    {
        return "不存在该用户！";
    }
}
```

(5) 这里使用重载创建了两个名称相同的 getInfo()方法。但是要注意，即使使用了 MessageName 属性为方法重命名，还是会出现如图 2-39 所示的错误。



图 2-39　相同方法名时错误提示

(6) 下面来解决这个问题。在新建 Web 服务时，系统自动生成的代码中有如下代码：

```
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
```

我们需要将它改成如下代码：

```
[WebServiceBinding(ConformsTo = WsiProfiles.None)]
```

(7) 这样，再次运行就不会出现同样的错误了。

2.6.4　运行结果

现在，让我们运行 Web 服务看一下运行效果。在浏览器中输入 Web 服务 myWebService.asmx 所在地址，如图 2-40 所示。

可以看到，现在虽然两个方法名相同，但是已经没有错误了。而且显示了使用 MessageName 属性重命名的方法名以及 Description 属性描述 Web 方法的信息。

这里我们先对有参数的 getInfo()方法进行测试。单击该链接在进入页面的 name 文本框中输入测试值，再单击【调用】按钮，如图 2-41 所示。



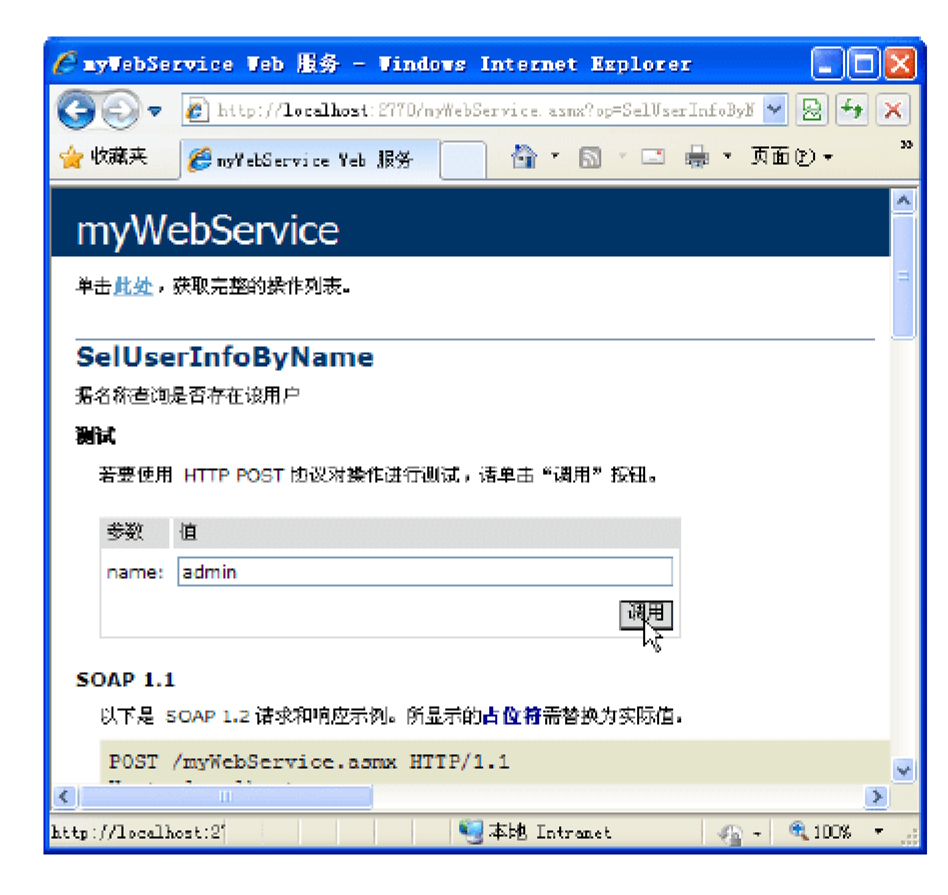




图 2-40　运行 Web 服务　　　　图 2-41　调用有参 getInfo()方法

此时，该方法将对数据库进行查询，根据查询结果显示是否存在该用户，如图 2-42 所示。





图 2-42　查询用户是否存在

现在，开始测试无参的 getInfo()方法。该方法主要是在 Web 中开启会话状态，用户前 3 次调用 Web 方法时返回的结果如图 2-43 所示；调用次数超过 4 次的时候，程序将返回不同结果，如图 2-44 所示。





图 2-43　前 3 次访问　　　　图 2-44　第 4 次访问

2.6.5　实例分析



源码解析：

在本实例中，我们创建了两个同名的 Web 服务方法，并使用 WebMethod 的 MessageName 属性进行了重命名。在第一个方法中使用了 EnableSession 属性对 Web 服务方法启动会话状态。在第二个方法中使用了 TransactionOption 属性，使 Web 服务与数据库交互时具有事务功能。

最后，提示一下，本实例所用的 LoginInfo 表中共有三列，分别是 Id、Name、Password。

2.7　常见问题解答

2.7.1　如何生成代理类，如何创建一个 Web 服务



如何生成代理类，如何创建一个 Web 服务？
网络课堂：http://bbs.itzcn.com/thread-15119-1-1.html

如题，新手不知问题如下：

(1)　如何用 wsdl 文件生成代理类的 cs 文件？

(2)　然后用这个代理类怎么做才能创建一个 Web 服务？

【解决办法】

(1)　如何用 wsdl 文件生成代理类 cs 文件？

答：本来要使用 wsdl 命令的，但如果有 Visual Studio 环境的话，添加 Web 引用即可。

例如，如下命令就会产生一个 xxx.cs 的 Web 服务代理类。

```
R:\>wsdl "http://localhost:1970/WebSite3/Service.asmx?WSDL"
Microsoft(R) Web Services 描述语言实用工具
[Microsoft (R) .NET Framework, Version 2.0.50727.1432]
Copyright (C) Microsoft Corporation. All rights reserved.
正在写入文件“R:\Service.cs”。
```

(2)　然后用这个代理类怎么做才能创建一个 Web 服务？

答：只有先创建一个 Web 服务，才能在客户端创建代理类来使用。具体方法是，添加对 Web 服务的引用，再实例化对象，然后调用其中的方法。

2.7.2　添加 Web 服务引用时的问题



添加 Web 服务引用时的问题？
网络课堂：http://bbs.itzcn.com/thread-15120-1-1.html

无法加载协议为“CMSserver.CMSWebServiceSoap”的终结点配置部分，因为找到了该协议的多个终结点配置。请按名称指示首选的终结点配置部分。

添加 Web 服务引用时出现上面的提示，这是怎么回事啊？盼高手给出解答。

【解决办法】

很简单，这是因为在添加对 CMSserver 的引用时，客户端的配置文件中出现了多个 endPoint 节点造成的。

解决办法就是在引用服务的项目中删除原来的服务，然后重新引用一次。

2.7.3 如何调试 Web 服务



如何调试 Web 服务？
网络课堂：http://bbs.itzcn.com/thread-15121-1-1.html

我在客户端引用了 Web 服务，想启动客户端程序调试代码，并跟踪到 Web 服务接口内部。请问应该如何做？麻烦详细说明一下！

Web Service 是在我本机的程序！

【解决办法】

可以尝试如下的步骤。

(1) 在解决方案中选中 Web 服务项目的“网站”菜单。

(2) 选择【ASP.NET 配置】，弹出属性设置的页面。

(3) 选中【应用程序】页，找到“调试和跟踪”一栏。

(4) 打开【配置调试和跟踪】选项卡。

(5) 在【配置调试和跟踪应用程序的设置】下选中【启用调试】复选框。

(6) 设置断点运行或按 F11 键运行，然后就可以单步调试 Web 服务程序的代码了。

2.7.4 Web Service 为什么没有 url 属性



Web Service 为什么没有 url 属性？
网络课堂：http://bbs.itzcn.com/thread-15122-1-1.html

请问各位，在用 C#编写 Web Service 时继承的是哪个类？为什么我的 Web Service 没有 url 属性，我用的是 Visual Studio 自动生成的代码。

【解决办法】

使用 Visual Studio 时只需要选择 Web Service 模板就行了，其他工作都由它来完成。当然，你也可以自己手动创建一个 Web Service。大致来说，需要满足以下几点要求。

(1) 使用的 WebService 类必须继承自 WebServices 类。

(2) 引入 WebServices 类所在的命名空间 System.Web.Service。

(3) 使用 WebService 指令声明当前为一个 Web Service。

(4) 使用“[WebMethod]”为类添加 Web 服务方法。

(5) 使用.asmx 为扩展名保存 Web 服务文件。

例如，下面给出一个简单得不能再简单的例子。

```
<%@ WebService Language="C#" Class="TestWebService " %>
using System.Web.Services;
public class TestWebService:WebServices
{
    [WebMethod]
```

```
    public string Hello()
    {
        return "Hello";
    }
}
```

2.7.5　WebMethod 和 WebMethod()有什么区别



请问 WebMethod 和 WebMethod()有什么区别?
网络课堂：http://bbs.itzcn.com/thread-15123-1-1.html

刚刚接触 Web 服务，看到有些带括号，有些不带的，有什么区别呢？
比如：

```
 [WebMethod]
public string HelloWorld()
{
return "Hello World";
}
[WebMethod()]
public DataSet GetACCData(string TBName, string tiaojian1)
{
    string x = Server.MapPath("person.mdb");
    string f = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + x;
    OleDbConnection ACConn = new OleDbConnection(f);
    DataSet DS = new DataSet();
    string k = "SELECT * From " + TBName + " where 姓名 like '%" + tiaojian1 +
    "%'";
    OleDbDataAdapter NComm = new OleDbDataAdapter(k, ACConn);
    NComm.Fill(DS, TBName);
    return DS;
}
```

【解决办法】

在 Web 服务中，每个要对外公布的方法名上面必须加一个 WebMethod，而在括号里面是给 WebMethod 的各个属性赋值的！

WebMethod 有 6 个属性：Description、EnableSession、MessageName、TransactionOption、CacheDuration 和 BufferResponse。

例如“WebMethod(EnableSession = true)”就表示在这个 WebMethod 方法中可以访问 Session 中的值。默认情况下 WebMethod 是禁用 Session 的！

2.8　习　　题

一、填空题

(1) Web 服务方法需要添加__________标签才能调用。

(2) 为了方便其他程序进行代理，我们需要将创建好的 C#文件编译成__________文件。

(3) 我们可以通过__________实用程序为 Web 服务创建代理类。

(4) WebService 的 Description 属性用于向 Web 服务添加描述信息，它被放到了__________文档里。

(5) 有一个使用 C#编写的 Web 服务位于 http://localhost/test.asmx，可以使用__________命令生成代理。

```
wsdl /l:cs /n:aSpace http://localhost/test.asmx
```

(6) 使用记事本创建一个 Web 服务必须继承__________类，它位于__________命名空间。

二、选择题

(1) ASP.NET Web 服务存在于扩展名为__________的文件中。

A. .asmx　B. .aspx　C. .ascx　D. .dll

(2) 在用户创建代理前，应当确保系统中安装了某些实用程序，下列__________不是必须的。

A. wsdl.exe　B. vbc.exe　C. vab.exe　D. csc.exe

(3) 下面不是 WebService 处理指令可以使用的属性是__________。

A. WebMethod　B. Language

C. Codebehind　D. Class

(4) 以下代码块中，需要在空格处填写的代码是__________。

```
[WebMethod(__________)]
public string  getCurrentUsers() {
    if (Session["users"] == null)
    {
        return "";
    }
    else
    {
        return Session["users"].ToString();
    }
}
```

A. MessageName = getCurrentUsers

B. CacheDuration=60

C. EnableSession=true

D. BufferResponse = true

(5) 在一个 Web 服务类中，下面__________方法能够被正确调用。

A.

```
    [WebMethod]
    public string Hello()
    {
        return "Hello";
```

```
}
```

B.

```
[WebMethod]
public static string Hello()
{
    return "Hello";
}
```

C.

```
public static string Hello()
{
    return "Hello";
}
```

D.

```
public string Hello()
{
    return "Hello";
}
```

(6) 下面不属于服务引用与 Web 引用的区别是__________。

A. 服务引用与 Web 引用都使用 wsdl 命令来生成代理

B. 在 Web 应用程序中可以同时引用服务和 Web

C. 窗体应用程序只有添加服务引用

D. 服务引用生成的代理只能被.NET 3.0 以上的客户端调用

(7) 使用 WebMethod 属性的 TransactionOption 选项可以为方法启用事务功能，下面不属于该选项值的是__________。

A. Disabled　　B. Enable　　C. NotSupported　　D. Supported

三、上机练习

上机练习 1： 模拟一个网上银行支付平台 Web 服务。

在本章中，我们主要学习了如何创建一个 Web 服务，和如何在项目中引用 Web 服务，也学习了 Web 服务的属性和 WebMethod 的属性。

本次上机练习中，我们将通过模拟一个网上银行支付平台，来进一步加深对 Web 服务的认识。效果图如图 2-45 所示。

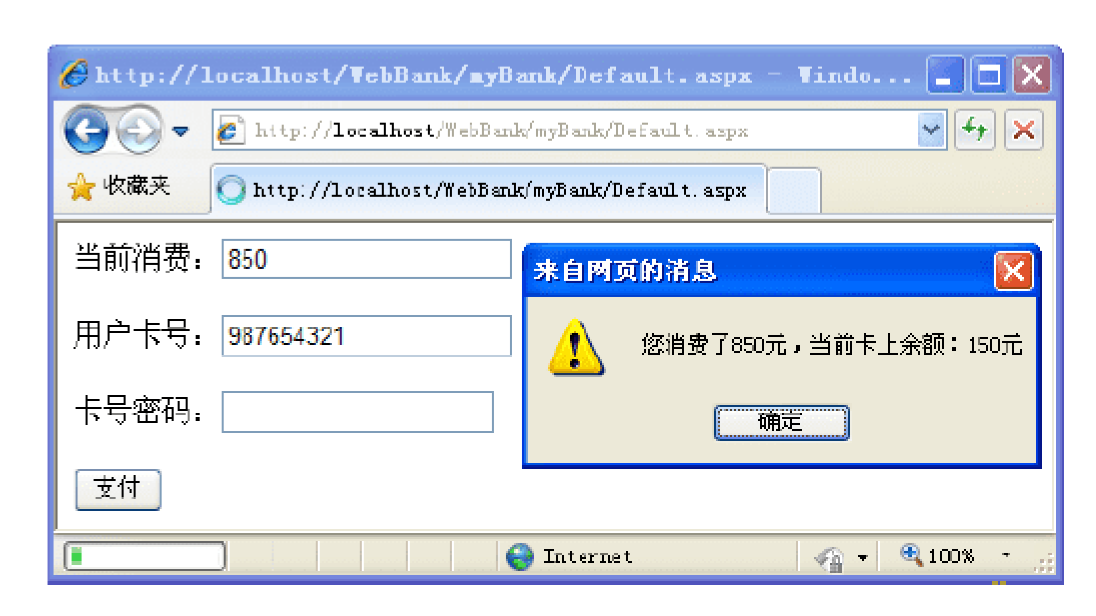


图 2-45　最终效果图



第 3 章　Web 服务基础知识之 XML 技术

内容摘要：

XML 是 Web 服务的基础之一，渗透到了 Web 服务的各个层次。也就是说如果没有 XML 就没有 Web 服务。在本书后面介绍 Web 服务的各种相关技术时，都会涉及 XML 及其相关术语，例如：XML 元素、属性、命名空间等。如果你对这些概念不够了解，加之于不能在开发过程中熟练地使用，那在以后学习 Web 服务时就会感到非常吃力和困惑。

因此本章将详细介绍 XML，包括 XML 的简介、XML 的声明、XML 的属性、XML 的命名空间、XML 的实体引用以及 DTD 等。

学习目标：

- 了解什么是 XML
- 掌握 XML 的标记与元素
- 熟悉 XML 中的属性
- 熟练掌握 XML 中命名空间的使用
- 掌握实体引用以及 CDATA 的使用
- 熟练使用文档类型定义 DTD

3.1　和我一起学 XML

提及 HTML，我想大家并不陌生，甚至会很自豪地说：我从事美工，对 HTML 的研究简直就到了炉火纯青的地步。但是，随着技术的发展，你有没有感觉到 HTML 语言的可扩展性不强呢？我想，答案是肯定的。眼下，长江后浪推前浪，专家怎能容忍 HTML 语言的局限性继续存在呢？因此就有了灵活性很高的 XML 标记语言。

下面我们来看一下 XML 的发展史。

XML 是从 1996 年开始有其雏形，并向 W3C(全球信息网联盟)提案，而在 1998 年 2 月发布为 W3C 的标准(XML1.0)。XML 的前身是 SGML(Standard Generalized Markup Language，标准通用标识语言)，是自 IBM 从 20 世纪 60 年代就开始发展的 GML(Generalized Markup Language，通用标识语言)标准化后的名称。

随着 Web 的广泛应用，W3C 逐渐意识到 HTML 的局限性，主要有以下几点。

- 不能解决所有解释数据的问题，例如：影音文件或化学公式、音乐符号等其他形态的内容。
- 效率问题，例如：需要下载整份文件，才能开始对文件做搜索的动作。
- 扩充性、弹性、易读性均不佳。

为了解决以上问题，专家们使用 SGML 为基础，并依照 HTML 的发展经验，制订出一套使用时规则严谨，但是描述简单的语言：XML。XML 是在这样的背景下诞生的——能否有一个更中立的方式描述简单的，让消费端自行决定要如何消化、呈现服务端提供的信息？

XML 被广泛用来作为跨平台之间交互数据的形式，主要针对数据的内容。通过不同的格式化描述手段(XSLT 或者 CSS)可以完成最终的形式表达(生成对应的 HTML、PDF 或者其他的文件格式)。XML 的目的在于提供一个对信息能够做精准描述的机制，从而弥补 HTML 太过于表现导向的不足。

下面我们就来具体介绍什么是 XML。



视频教学：光盘/videos/03/XML 简介.avi　　　　长度：6 分钟

XML(Extensible Markup Language，可扩展标记语言)与 HTML 一样，都是 SGML。与其他语言不同的是：XML 没有规定的标记，即用户可以根据需要自行创建符合 XML 规范的标记来描述 XML 文档要表达的内容。

XML 也是一种元标记语言，可用于定义其他与特定领域有关的、语义的、结构化的标记语言。

通过上面的介绍，我们了解到 XML 是一种可以自行创建的标记语言。用户自己创建标记与元素，不仅用户能够理解文档的内容，而且具有相当高的灵活性。另外 XML 标记的主要特征是将内容与其表示相分离。

上面提到 XML 允许用户创建标记与元素，但需要遵循文档规范。那么如何保证遵循 XML 的文档规范呢？很简单，每个 XML 都遵循一定的文档结构。下面我们就来详细介绍 XML 的文档结构。

1)　序言

序言是 XML 文档的开始，用来表示对 XML 数据进行编译的开始以及描述字符的编码方式。

XML 文档的序言主要包括 XML 的声明和注释两部分。XML 的声明用来说明该文档是一个 XML 文档，并且指定该文档使用的编码方式。注释则是用来说明该 XML 文档有什么特点，以便用户对文档理解的更加清晰、透彻。

2)　主体

主体即是 XML 文档用来描述数据的，通常由标记、元素和属性组成。

3.2　创建一个简单的 XML 文档

我是一个购物爱好者，看到比较漂亮而且实惠的商品我总是第一时间订购。对于经济并不富裕的我来说，总是偏爱于那些打折的商品，因此我时刻关注着某些商品的打折信息。不知道大家是否仔细观察过，在街道两边的商品每隔一段时间打折价都不相同，当时我就在想这么多商品，每件商品的打折都不相同，如果要一个一个地标记岂不是太麻烦了吗？以前恐怕是这样吧，但是现在我们可以变通一下，使用 XML 文档来标记这些商品的打折价是一个很不错的选择哦。

下面就以简单的 XML 文档来讲一下，如何使数据更易于扩展和更新。

**视频教学**：光盘/videos/03/创建一个 XML 文档.avi　　长度：12 分钟

3.2.1　基础知识——XML 的声明和注释

XML 的声明以“<?xml”开始，以“?>”结束。例如：

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
```

在上述代码中，“<?　?>”中的各个参数的含义如下。

- version：指定 XML 的版本号。通常情况下 Web 支持的是 XML 1.0 的版本。
- encoding：指定 XML 文档的编码格式。常见的有 UTF-8 和 GB2312 两种，默认为 UTF-8。
- standalone：指定使用 DTD 的形式，有两个值：yes 或 no。XML 文档的默认值是 no，则说明需要引用外部实体。否则说明所有必须的实体声明都包含在文档中。

XML 注释是对文档结构或者内容的解释，由于它不属于 XML 文档的内容，因此 XML 解析器不会处理。注释以“<!--”开始，以“-->”结束。其语法格式如下所示：

```
<!-- 需要注释的内容-->
```

注释主要可以用在文档的序言部分、文档的文本内容中以及文档之后。但是“-->”字符不能用在注释的内部，由于解析器一旦碰到“-->”就会看做一个注释的结束，而后面的内容则被作为普通的 XML 文档处理。对于其他合乎规范的 XML 字符均可出现在注释的内部。

3.2.2 基础知识——XML 的标记与元素

众所周知，在 HTML 语言中，我们使用预先定义好的标记来完成文档。例如：我们来做一位明星的履历。代码如下所示：

```
<dt>明星信息
<dd>我最喜爱的明星信息
  <ul>
    <li>A 明星</li>
    <li>男</li>
    <li>1.78m</li>
    <li>篮球、唱歌</li>
    <li>眼泪，是成长的代价。如哪天，你的泪不再流了，想说，那不是你大了，而是，你的心已经死
了</li>
  </ul>
</dd>
</dt>.
```

在上述代码中，我们使用的是 HTML 标记语言，而且只能使用预先定义好的标记来完成。但 XML 文档就不同了，我们可以自定义标记。同样定义明星的履历，可以写成如下所示的代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<明星>
  <名称>A 明星</名称>
  <性别>男</性别>
  <身高>1.78m</身高>
  <爱好>篮球、唱歌</爱好>
  <格言>眼泪，是成长的代价。如哪天，你的泪不再流了，想说，那不是你大了，而是，你的心已经死
了</格言>
</明星>
```

从上述代码中可以看出 XML 的标记是由“< >”来定义的。左边以尖括号“<”开始，右边以尖括号“>”结束，且中间含有数据的标记被称为非空标记。在上述 XML 文档中，“<名称>”和“</名称>”都是标记，“<名称>”、“<性别>”等被称为开始标记，“</名称>”、“</性别>”等被称为结束标记。



在 XML 文档中，标记都是成双成对出现的，有开始标记就少不了结束标记。

你可能会疑惑，既然有非空标记，那么是不是有空标记呢，答案是肯定的。

在 XML 文档中，标记可以分为非空标记和空标记。

下面来看一下空标记是如何定义的。

空标记是以尖括号“<”开始，以尖括号“/>”结束的标记。由于空标记不包含任何内容，因此空标记没有开始标记和结束标记。但空标记中可以包含属性，下面我们使用空标记来描述一下明星的履历。代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<明星 名称="A 明星" 性别="男" 身高="1.78m" 爱好="篮球、唱歌" 格言="眼泪，是成长的代价"
/>
```

在上述代码中，“姓名”、“性别”、“身高”、“爱好”等均是标记明星包含的属性。



这里提到了 XML 的属性，在本章后面将对它进行详细介绍。

通过上面对标记的介绍，我们了解了如何定义标记，下面就来介绍 XML 文档中的元素。

元素是 XML 文档中最重要的组件，用来描述此文档所包含的数据。在 XML 文档中有且只有一个根元素，其他元素是在根元素内部以树状的结构来显示的。

同样在 XML 文档中，元素可以分为非空元素和空元素两种类型。首先来看一下什么是非空元素。

所谓非空元素是由开始标记、结束标记以及两标记之间的数据构成的。而开始标记和结束标记用来描述标记之间的数据，所以说这个标记之间的数据就被称为元素的值。下面来看一下非空元素的语法格式，如下所示：

```
<开始标记>描述的数据(元素的值)</结束标记>
```

在前面我们学习了空标记，其实空标记和空元素的格式是一样的。在 XML 解析器中，程序对空元素和空标记的处理是相同的，因此二者的作用是等价的。下面来看一下空元素的定义吧。

所谓空元素就是不包含任何内容的元素。语法格式如下所示：

```
<开始标记></结束标记>
```

空元素的语法格式还可以这样写，如下所示：

```
<开始标记 />
```

为了更好地了解 XML 文档的元素，我们通过一个简单的 XML 文档来说明一下。代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<用户>
  <编号 ID="123" ></编号>
  <用户名 >dcy</用户名>
  <身高 >1.67m</身高>
  <爱好 >爱的供养</爱好>
</用户>
```

在该文档中，“用户”是文档的根元素，根元素内部有 4 个子元素，分别是：编号、用户名、身高和爱好。其中“<编号 ID="123" ></编号>”是一个空元素，而其他的三个子元素均是非空元素。当然“<编号 ID="123" ></编号>”空元素还可以这样写，代码如下所示：

```
  <编号 ID="123"  />
```

瞧，这样是不是简化了 XML 文档啊。

俗话说“没有规矩不成方圆”，在任何编程语言中都有特定的语言规范，那么同样在 XML 文档中为了更能准确地描述数据，也有一套命名规则。下面我们来看一下 XML 标记和元素的命名规则，如下所示。

- 标记或元素名称须以字母、下划线“_”或者中文开头，不能以数字或者标点符号开头。例如：<_name>、<name>、<名称_name>等命名都是正确的。
- 元素名中可以包含字母、数字及其他字符，如<name>、<用户名>、<ab123>等。但并不是所有的软件都能很好地支持中文的命名，所以尽量使用英文来命名。在 XML 文档中，不要使用“:”来命名，此字符会在 XML 命名空间中用到。
- 元素名称不能以 XML 的任意形式开头(如 XML 或者 Xml 等)。例如：<xmlBook>、<XMLbook>、<XmlBook>等。
- 名称中不能包含空格，例如：<abc　abc>。
- 文档中标记必须对应，也就是说，在 XML 文档中只要有开始标记，就必须有结束标记。
- 标记区分大小写，例如<name>和<Name>是两个完全不同的标记。
- 元素标记必须要合理地进行嵌套。

3.2.3 实例描述

我朋友家是开超市的，我去那里帮忙，恰巧碰到厂家来送货，朋友交代我将这些商品的价格记录下来，并且将记录下来的商品分一下类。这样我就一直在那里记录，但是不久就把我自己给搞晕了，这样的记录连我自己都看不明白，更别提别人了。突然间我想能不能使用 XML 文档来将这些商品进行总体分类呢，然后将属于某一类的商品作为其类的子元素。

因为 XML 的可扩展性很高，并且很容易维护和修改。这样想过之后，我就马上行动起来了，很快我就将这些商品分类成功了。我不得不惊叹学以致用真得很重要。

下面就来看一下我的实现方案。

3.2.4 实例应用

【例 3-1】 创建一个简单的 XML 文档

(1) 首先创建一个名称为 GoodsXml 的 XML 文档。

(2) 在 GoodsXml 文件中添加如下代码所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<goods>
  <SkinCare>
    <anti_wrinkle>
      <SkinName>真奢华</SkinName>
      <SkinPrice>450 元</SkinPrice>
      <SkinEffect>去干纹</SkinEffect>
    </anti_wrinkle>
```

```
    <hydrating>
      <SkinName>dermare</SkinName>
      <SkinPrice>250 元</SkinPrice>
      <SkinEffect>双因子补水维护精华</SkinEffect>
    </hydrating>
    <whitening >
      <SkinName>美白</SkinName>
      <SkinPrice>270 元</SkinPrice>
      <SkinEffect>美白维护精华</SkinEffect>
    </whitening>
  </SkinCare>
  <shampoo>
    <repair >
      <shampooName>潘婷</shampooName>
      <shampooPrice>50 元</shampooPrice>
      <shampooEffect>修复干枯洗发露</shampooEffect>
    </repair>
    <Prevent>
      <shampooName>迪彩</shampooName>
      <shampooPrice>30 元</shampooPrice>
      <shampooEffect>防干枯洗发露</shampooEffect>
    </Prevent>
  </shampoo>
</goods>
```

(3)　保存上面对文件的修改。

3.2.5　运行结果

运行程序，显示的效果如图 3-1 所示。

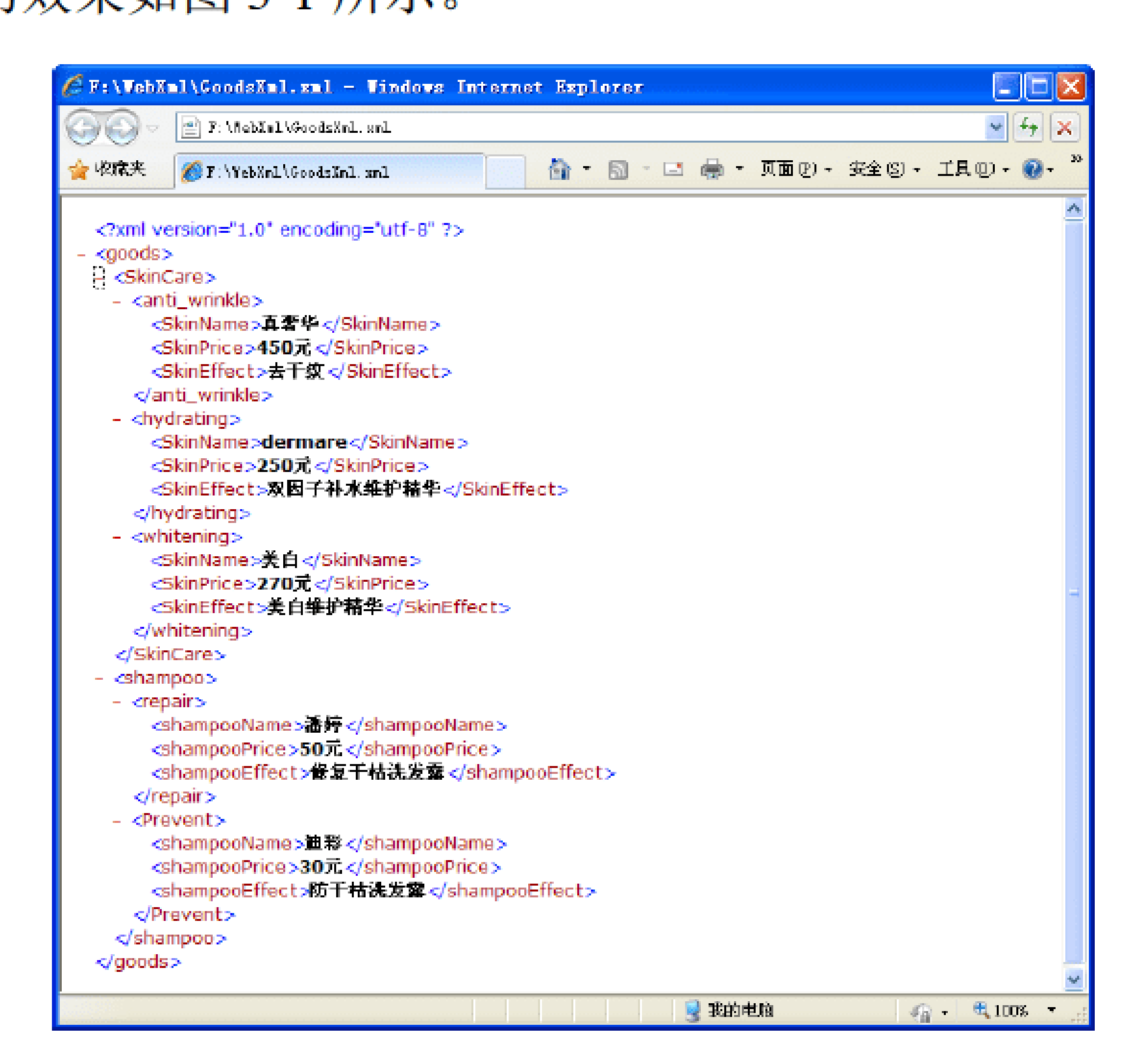


图 3-1　商品的分类效果

3.2.6 实例分析



源码解析：

在本实例中，我们使用商品 goods 作为此文档的根元素，以护肤品 SkinCare 和洗发露 shampoo 作为此根元素的子元素。接着 anti_wrinkle、hydrating 和 whitening 则是子元素 SkinCare 下的嵌套子元素，而 repair 和 Prevent 则是子元素 shampoo 下的商品。瞧，这样是不是很容易就能增加商品的分类或者修改商品的价格。

3.3 XML 的属性

学过 HTML 语言的人都知道，要想为预先定义好的标记设置样式时，可以使用内嵌样式或者外部样式。例如：使用 table 标签创建一个背景色为红色、边框为 1 像素、边框颜色为蓝色的表格。如果使用内嵌样式，就会用到此标签中的 bgcolor、border、bordercolor 等属性。那么在 XML 标记语言中有没有类似的属性呢，答案是肯定的。

下面就来看一下 XML 的属性。



视频教学： 光盘/videos/03/我的属性.avi　　长度：6 分钟

XML 属性是将一些额外的信息附加到元素上，从而使文档对元素数据描述得更加具体。如果用户不喜欢通过子元素来描述元素的一些特性，那么使用属性来存储会是一个不错的选择。属性一般在开始标记中声明，由属性名和值构成。下面来看一下如何在非空元素中添加属性，语法如下所示：

```
<标记名 属性列表>描述的数据 </标记名>
```

在空元素中添加属性，语法如下所示：

```
<标记名 属性列表> </标记名>或者<标记名 属性列表/>
```

下面我们使用属性来描述一下今天的公交车情况，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<公交车>
  <车 95 长="4m" 宽="3m" 数量="40 人" >拥挤</车 95>
  <车 K6 长="4m" 宽="3m" 数量="20 人" > </车 K6>
</公交车>
```

在非空子元素“车 95”中，属性长、宽、数量描述了此路车的拥挤状况。而在空子元素“车 K6”中，属性长、宽、数量描述了此路车的宽松状况。

在 XML 中，标记和元素有命名规则，那么属性是不是也需要遵循一些命名规范呢，下面我们就来看一下。

- 属性名的命名规则和元素的命名规则相同，可以由字母、数字、中文以及下划线组成，但必须以字母、中文或者下划线开头。
- 属性名同样也区分大小写。
- 属性值必须使用单引号或者双引号。例如：“name”和‘name’描述的是相同的属性值。
- 如果属性值中需要使用左尖括号“<”、右尖括号“>”、连接符号“&”、单引号“’”或者双引号“””时，必须使用实体引用。



在这里提到的实体引用，在本章后面将对它进行详细介绍。

在 XML 文档中，使用子元素和属性都可以实现对数据描述信息的存储。下面我们使用子元素来描述该文档的数据信息，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<公交车>
  <车 95>
    <长>4m</长>
    <宽>3m</宽>
    <数量>40 人</数量>
  </车 95>
  <车 K6>
    <长>4m</长>
    <宽>3m</宽>
    <数量>20 人</数量>
  </车 K6>
</公交车>
```

在 XML 文档中，使用属性和子元素均可以描述数据信息，只是根据执行环境和自己的喜好来决定使用哪种方式而已。在文档中使用属性，有以下缺陷：

- 属性不容易扩展。
- 属性不能描述文档结构。
- 属性很难被程序代码处理。
- 属性值很难通过 DTD 进行测试。

综合以上几点来看，在文档中使用属性还是有很大的局限性。由于 XML 的可扩展性高，因此数据在 XML 中需要经常添加或者修改，如果将这些数据放入属性中存储，那么很容易想象，这将会对数据的维护和更新造成多大的麻烦。

3.4 展示个性的名言警句

我们在使用.NET 做网站或者系统时，定义了很多相同的属性，但是程序在编译的时候却能准确无误地执行，为什么？这是因为这些相同的属性有不同的命名空间，正是声明了这些命名空间，才能使程序调用这些属性时不发生混乱。那么，在 XML 文档中同样存在命名空间这

层含义。下面我们来看一下 XML 文档中的命名空间。

 **视频教学**：光盘/videos/03/XML 命名空间.avi　　长度：9 分钟

3.4.1 基础知识——XML 命名空间

由于 XML 允许用户根据需要自定义标记，因此不可避免地会产生相同的标记。当解析器遇到相同的标记时就会产生混乱。而此时的 XML 命名空间正是解决这些混乱的关键，XML 命名空间是 XML 文档中所使用元素或属性名称的集合，它在元素或者属性名称上加以限定，避免名称相同的元素或者属性发生冲突。

命名空间通常由统一资源标识符(Uniform Resource Identifier，URI)确定，在 XML 文档中作为元素类型或者属性名称使用。另外统一资源定位符(Uniform Resource Locator，URL)、统一资源名称(Uniform Resource Name，URN)、全局唯一标识符(Uniform Unique Identifier，UUID)和全球唯一标识符(Globally Unique Identifier，GUID)等均可以作为命名空间来使用。

既然 XML 命名空间有如此大的作用，那么下面我们就来看一下如何声明命名空间。

1．XML 中命名空间的声明

XML 命名空间是由前缀和本地部分组成的，中间用冒号“:”隔开，前缀标识元素或属性所在的名称空间，本地部分标识名称空间中的某个元素或属性。XML 命名空间的声明语法格式如下所示：

```
xmlns:prefix="URI"
```

上述语法中各参数的含义如下所示。

- xmlns：必需的属性。
- prefix：命名空间的别名，它的值不能为 xml。
- URI：用来标识抽象资源，按照 URI 规范，有两种常规类型的 URI 分别是：统一资源定位符 URL、统一资源名称 URN。这两种类型的 URI 都可以确保命名空间的唯一性。

下面我们通过一个例子来说明如何声明和使用命名空间，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<goods xmlns:info="http://www.qxian.com/SkinCare"
xmlns:shapoo="urn:qxian.shampoo">
  <info.SkinCare>
    <info.anti_wrinkle>
        <info.SkinName>欧莱雅</info.SkinName>
        <info.SkinPrice>450 元</info.SkinPrice>
        <info.SkinEffect>去干纹</info.SkinEffect>
    </info.anti_wrinkle>
  </info.SkinCare>
  <shapoo.shampoo>
    <shapoo.repair >
        <shapoo.shampooName>拉芳</shapoo.shampooName>
        <shapoo.shampooPrice>50 元</shapoo.shampooPrice>
        <shapoo.shampooEffect>修复干枯洗发露</shapoo.shampooEffect>
```

```
    </shapoo.repair>
    <shapoo.Prevent>
        <shapoo.shampooName>迪彩</shapoo.shampooName>
        <shapoo.shampooPrice>30 元</shapoo.shampooPrice>
        <shapoo.shampooEffect>防干枯洗发露</shapoo.shampooEffect>
    </shapoo.Prevent>
  </shapoo.shampoo>
</goods>
```

在上述代码中，我们声明了两个命名空间 info 和 shapoo，并且虚拟了两个命名空间的 URI。此命名空间中的元素或者属性名称需要使用“命名空间”来引用命名空间中的子元素。在 info 命名空间中，描述了两个子元素 SkinCare 和 anti_wrinkle；而在 shapoo 命名空间中，描述了两个子元素 repair 和 Prevent。由于解析器不对 URI 进行访问，因此对 XML 文档的运行结果没有影响。

程序运行的结果如图 3-2 所示。

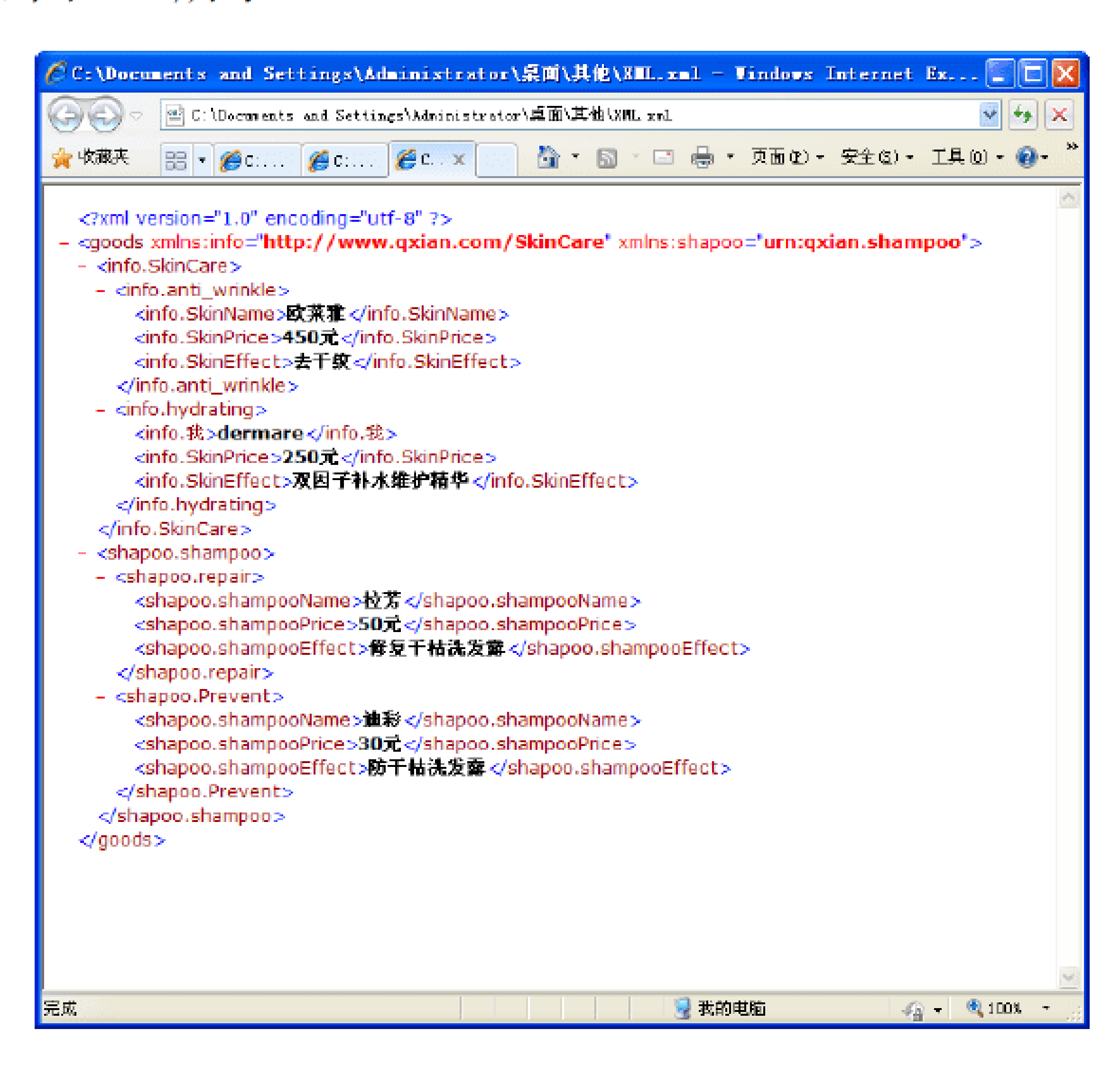


图 3-2　使用两个命名空间的效果

另外，在 XML 文档中还有一个空间被称为默认命名空间。所谓默认命名空间是说没有为命名空间定义别名，格式如下所示：

```
xmlns="URI"
```

下面我们来看一个默认命名空间的例子，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<weather xmlns="http://www.weather/weather">
  <direction>微风</direction>
  <temper>15 度</temper>
  <state>晴转多云</state>
</weather>
```

在上述文档中，我们定义了一个天气根元素 weather，并且声明了一个默认的命名空间，

而在此元素下的 direction、temper 和 state 属性都是引用这个命名空间。

因为 XML 文档中只能定义一个根元素，在根元素中声明了一个命名空间，说明此元素以及在其内部的子元素均可以使用此命名空间。但是如果我们在子元素中重新声明一个命名空间，那么在其子元素中的属性如何引用命名空间呢，这就需要我们了解命名空间的范围了。下面来讲解一下命名空间的范围。

2．命名空间的范围

和其他编程语言中的命名空间相似，命名空间都有其作用范围。在 XML 文档中，命名空间都是在某个元素内声明的，那么它的作用范围就是该元素。因此该元素以及所有子元素和属性均可以引用这个命名空间，而不属于该元素范围之内的子元素或者属性则需要重新声明此命名空间方可引用。

下面我们来看一个例子，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<lvyou xmlns:lvyouS="urn:lvyou.lvyou">
  <lvyouS.spring>
    <lvyouS.name>国际椰子节</lvyouS.name>
    <lvyouS.site>海南省海口市、文昌县、通什市</lvyouS.site>
    <lvyouS.attractions>举办椰城灯会、椰子一条街、黎族苗族联欢节、国际龙舟赛民族武术擂台赛。文体表演、黎族苗族婚礼、祭祖</lvyouS.attractions>
  </lvyouS.spring>
  <summer xmlns:summerS="http://www.lvyou/summer">
    <summerS.name>九寨沟</summerS.name>
    <summerS.site>九寨沟</summerS.site>
    <summerS.attractions>九寨沟以绝天下的原始、神秘而闻名。</summerS.attractions>
  </summer>
  <lvyouS.autumn>
    <lvyouS.name>阳光灿烂海南游</lvyouS.name>
    <lvyouS.site>阳光灿烂海南游</lvyouS.site>
    <lvyouS.attractions>海南气候宜人，冬可避寒，且自然风光优美，海湾波平浪静，柔软的沙滩洁白如银</lvyouS.attractions>
  </lvyouS.autumn>
</lvyou>
```

在上述文档中，我们在根元素 lvyou 内声明了一个命名空间 lvyouS，那么在此元素下的子元素 spring、summer、autumn 均可引用此命名空间 lvyouS。但是在子元素 summer 内部重新声明了一个命名空间 summerS，那么在子元素 summer 下的属性 name、site、attractions 则属于 summerS 命名空间。而其他的子元素或者属性不属于 summerS，因而不能引用此命名空间。

3.4.2 实例描述

回顾以往，感慨颇多，但是时代的竞争与发展由不得我们沉湎过去，唯一能做的就是好好把握今天，以便能使未来的生活更加充实、精彩。

为此，我使用 XML 文档中的命名空间把对昨天、今天和明天的生活慨叹用诗句记录下来，以便时刻提醒自己不要造成“少壮不努力，老大徒伤悲”的局面。

3.4.3　实例应用

【例 3-2】 展示个性的名言警句

(1)　创建名称为 future.xml 的文件。

(2)　在 future.xml 中添加如下代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<time xmlns="http://www.myTime/mytime">
  <yesterday xmlns:myyesterday="http://www.myTime/myyesterday">
    <myyesterday.name>金缕衣</myyesterday.name>
    <myyesterday.author>杜秋娘</myyesterday.author>
    <myyesterday.shiju>劝君须惜少年时。
      劝君莫惜金缕衣，
      有花堪折直须折，
      莫待无花空折枝。</myyesterday.shiju>
  </yesterday>
  <today xmlns:mytoday="http://www.myTime/mytoday">
    <mytoday.name>明日歌</mytoday.name>
    <mytoday.author>钱鹤滩</mytoday.author>
    <mytoday.shiju>明日复明日，
      明日何其多。
      我生待明日，
      万事成蹉跎。
      世人苦被明日累，
      春去秋来老将至。
      朝看水东流，
      暮看日西坠。
      百年明日能几何？
      请君听我明日歌!</mytoday.shiju>
  </today>
  <tomorrow xmlns="http://www.myTime/mytomorrow">
    <name>英</name>
    <author>泰戈尔</author>
    <shiju>生如夏花之绚烂，死如秋叶之静美</shiju>
  </tomorrow>
</time>
```

(3)　保存修改好的代码。

3.4.4　运行结果

运行程序，执行的结果如图 3-3 所示。

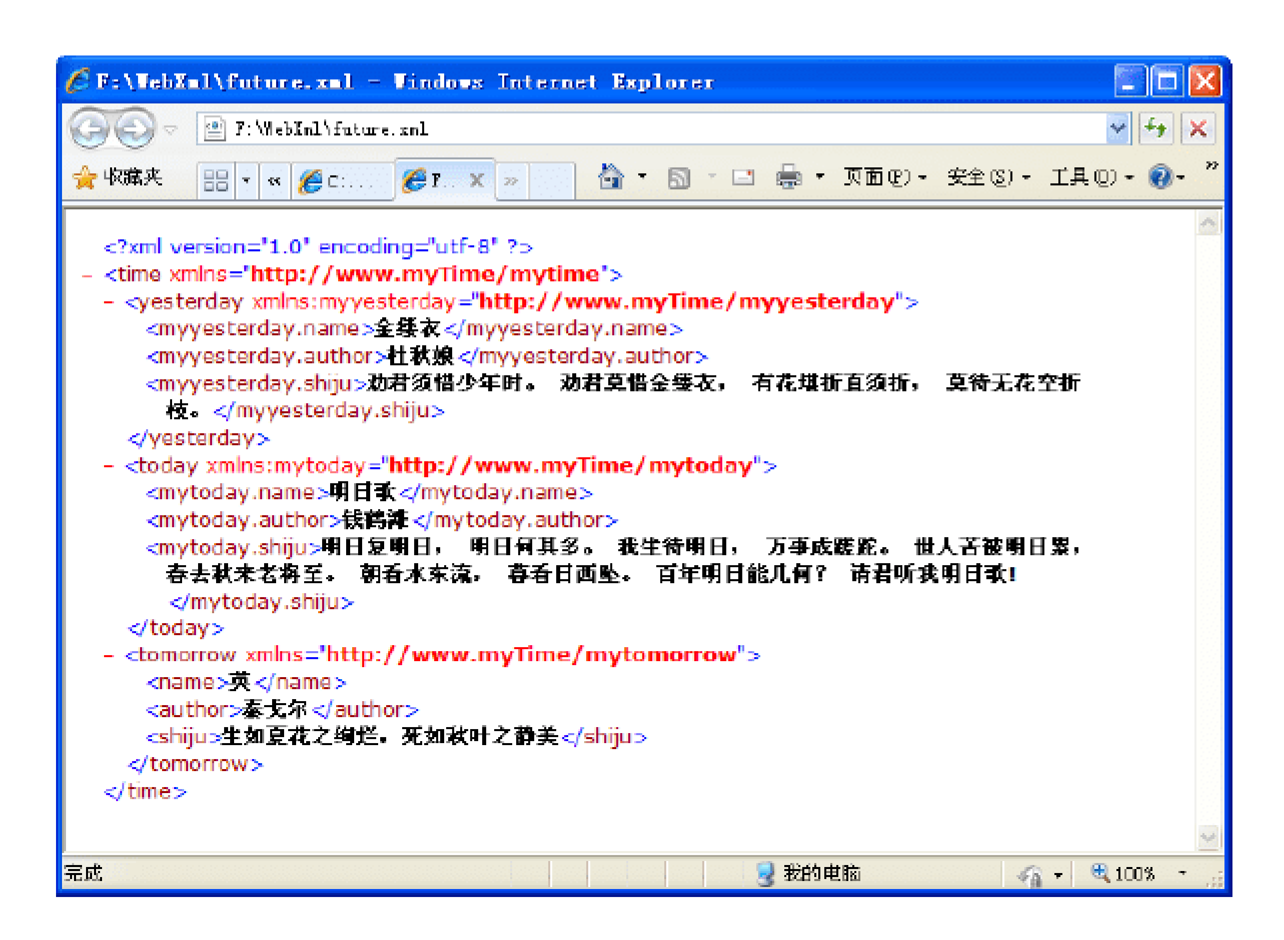


图 3-3　嵌套命名空间的效果

3.4.5　实例分析



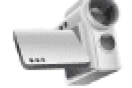
源码解析：

在该文档中，我们定义了一个根元素，并在根元素内声明了一个默认的命名空间。那么在此根元素下的子元素以及属性都属于这个默认的命名空间。在子元素 yesterday 和 today 中分别声明了命名空间 myyesterday 和 mytoday。

接着在子元素 tomorrow 中声明了一个默认的命名空间，那么在该子元素下的属性即属于该默认的命名空间。

3.5　歌　词　秀

我喜欢听音乐，并且热衷于 LRC 歌词，由于 LRC 歌词上有时间说明，对于我这个丝毫不懂音律的人来说有很大的帮助。俗话说，没有最好，只有更好，我一直想要设计出带自己特色的歌词模式。而恰巧这一节讲的是实体引用和 CDATA 区，因此我想从这里得到一些灵感，我迫不及待地想来看一下。



视频教学：光盘/videos/03/歌词秀.avi　　　　**长度：**7 分钟

3.5.1　基础知识——字符和实体引用

字符和实体引用是指可以向 XML 文档中引入其他信息，而不需要在文档中输入。例如在 XML 文档中会遇到大于号“＞”、小于号“＜”以及连接符号“＆”等，如果你想要将这些字符在 XML 文档中显示出来，那么就要考虑实体引用了。

实体引用是以“&”开始，以“;”结尾的引用。

实体引用的作用就是，当字符或数据中需要使用到这些特殊符号时，可以采用它的实体引用来代替。下面我们来看一下哪些特殊符号可以使用实体引用，如表 3-1 所示。

表 3-1　XML 的实体引用

实体引用	特殊字符	含　义
<	<	小于号
>	>	大于号
&	&	连接符
'	‘	单引号
"	“	双引号

看了表 3-1 中 XML 的实体引用，是不是感觉很不可思议，下面我们通过一个例子来说明一下。代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<生活>
  <名称>
    &lt;老男孩&gt;
  </名称>
  <样貌>
    &apos;生活像一把无情刻刀，&apos;    &amp;
    &quot;改变了我们模样&quot;
  </样貌>
</生活>
```

在上述文档中，我们使用了<、>、&、'、"等字符，那么程序运行时，解析器会将这些特殊字符解析成“<”、“>”、“&”、“’”、“““”。下面我们来看一下运行的结果，是否与我们推测的一致，如图 3-4 所示。

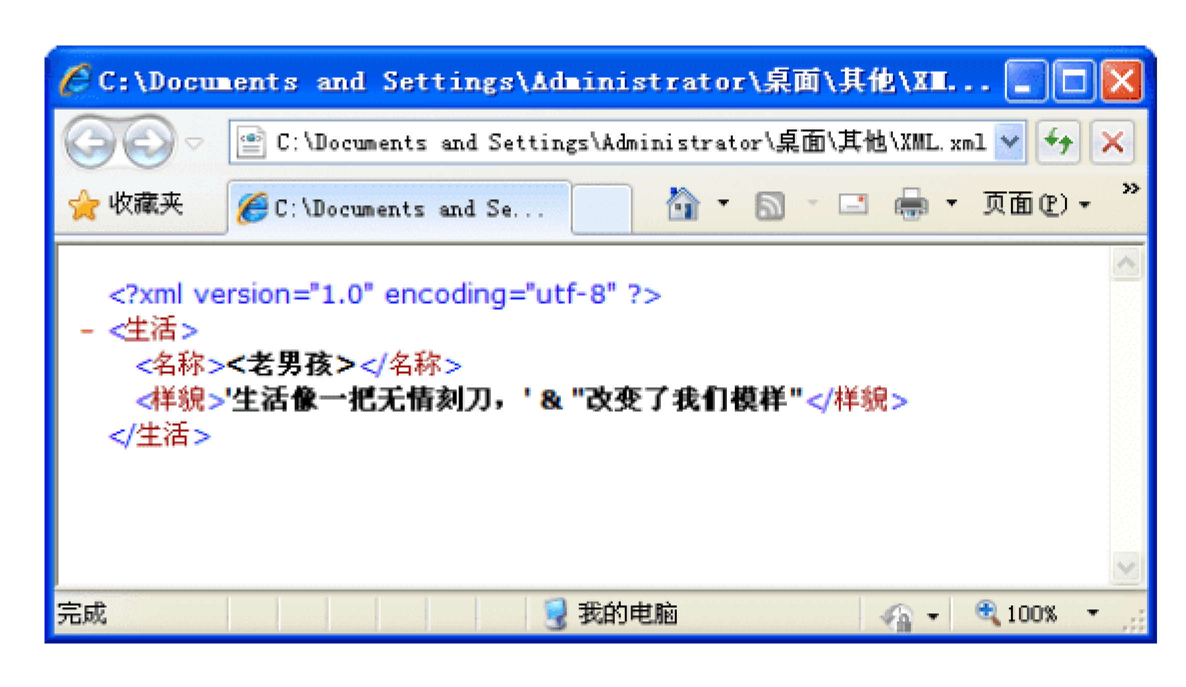


图 3-4　采用实体引用代替的效果

3.5.2　基础知识——CDATA 的使用

上面的字符和实体引用挺容易理解的，但是这个 CDATA 看上去挺难的，都有点胆怯了。

但不是有这样一句话吗，“世上所有的问题，就怕认真二字”，那么这次就让我们认真对待看上去挺难理解的 CDATA 区吧！

CDATA 区是用来包含文本的方法，通常用于建立代码的脚本，例如：JavaScript 脚本。放在 CDATA 区的内容会被 XML 解析器忽略，然后将会被 XML 处理程序当作字符数据来处理。下面我们来看一下使用 CDATA 的语法格式，如下所示：

```
<![CDATA[content]]>
```

了解了 CDATA 的语法格式，我们通过一个小例子来加深理解，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<company>
  <employee>
    <name>dcy</name>
    <department>.NET 编程</department>
    <position>公司小员工</position>
    <hobby>看电视</hobby>
    <hate>自私自利</hate>
  </employee>
</company>
```

运行程序，执行的结果如图 3-5 所示。

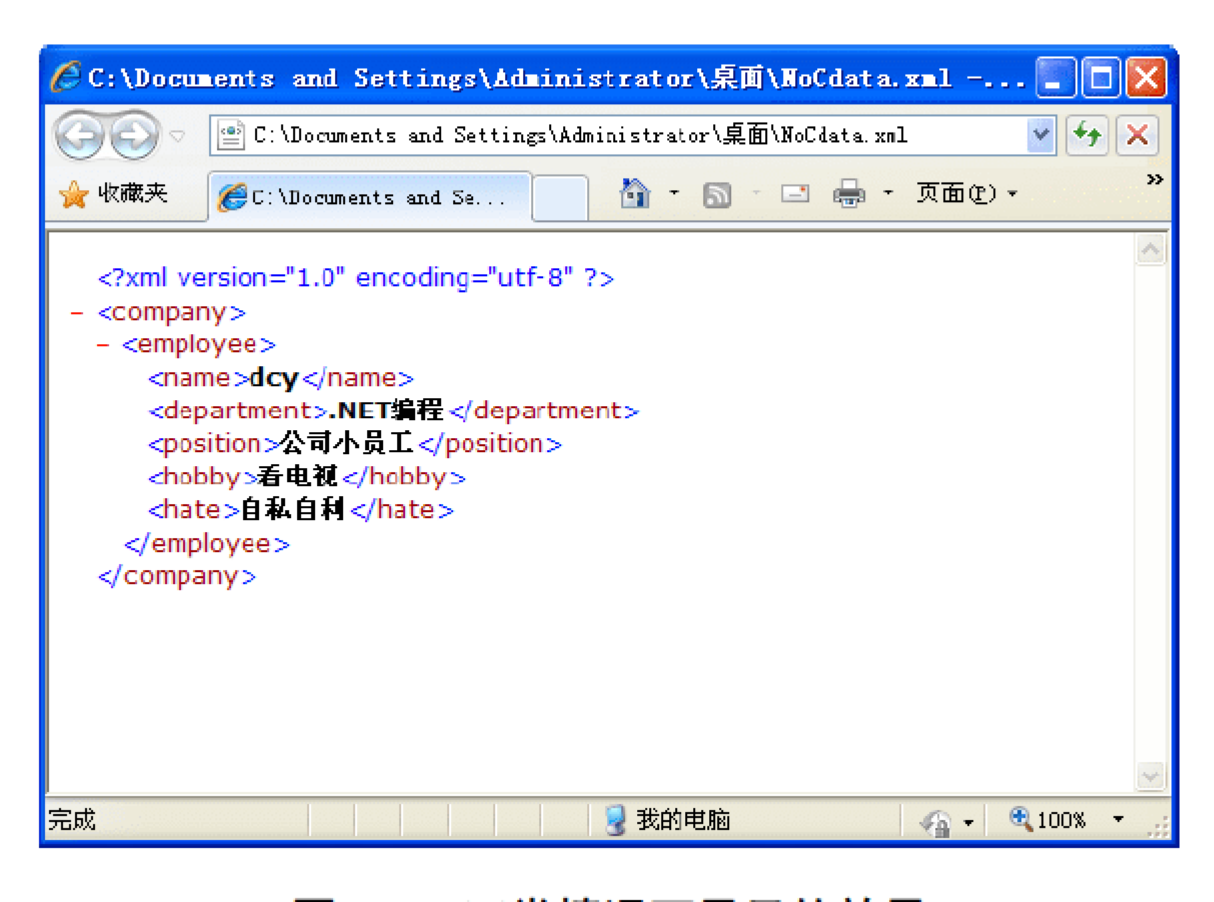


图 3-5　正常情况下显示的效果

在图 3-5 中，如果我不想让 XML 解析器解析子元素中的属性 hobby 和 hate，那么最理想的做法就是将这两个属性放到 CDATA 区中，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<company>
  <employee>
    <name>dcy</name>
    <department >.NET 编程</department >
    <position>公司小员工</position>
    <![CDATA[
    < hobby >看电视</ hobby >
    <hate>自私自利</hate>
```

```
   ]]>
  </employee>
</company>
```

保存修改好的代码。运行程序，得到的结果如图 3-6 所示。

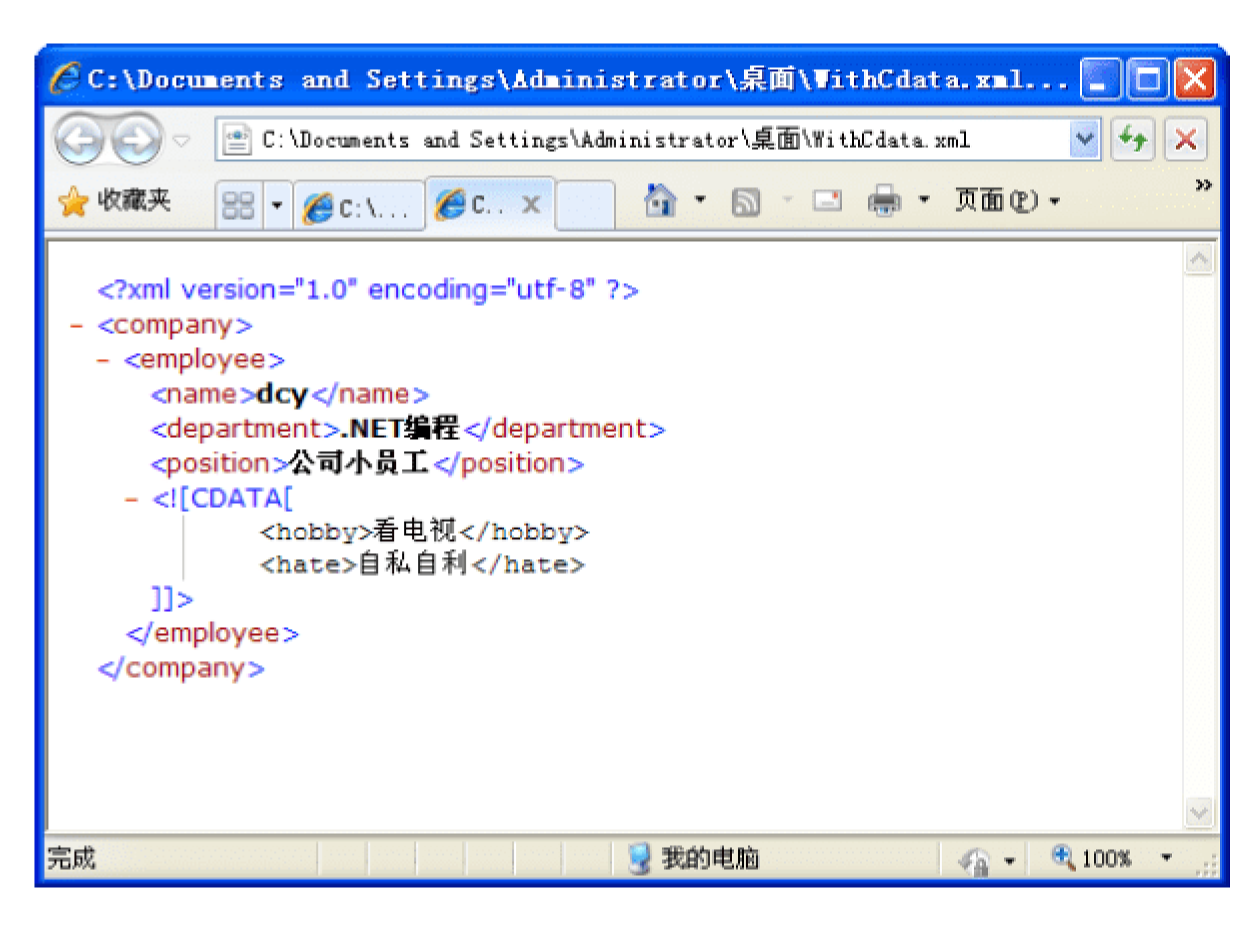


图 3-6　使用 CDATA 区的效果

在上面我们提到 XML 处理程序时会将 CDATA 区中的文本内容作为字符数据来处理，因此当文本文档大量地使用特殊符号时，我们就可以编写如下代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<歌词>
  <![CDATA[
  <名称>
    <爱的供养>
  </名称>
  <语句>
    '我用尽一生一世来将你供养，只盼望能停住你流转的目光，'    &
    "请赐予我无限爱与被爱的力量，让我能安心在菩提下静静的观想"
  </语句>
  ]]>
</歌词>
```

瞧，这样是不是比那些通篇文档都用实体引用代替简便多了。

3.5.3　实例描述

最近，我迷上了一首歌，名字是“一剪梅”，记得曾经有人说过：我们之所以留恋某个地方都是因为我们留恋那个地方的人。那么我们喜欢上一首歌也是因为我们被电视剧里面的情节所打动。为了留住这感动的时刻，我想用所学的 XML 字符和实体引用以及 CDATA 的知识将这首歌的歌词，以自己的方式展示出来。

下面来看我的实现方案。

3.5.4　实例应用

【例 3-3】 歌词秀

(1) 创建一个名称为 song.xml 的文件。

(2) 在 song.xml 文件中添加如下代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<geci>
  <quming> 一剪梅</quming>
  <yuju>
  &lt;00:03.00&gt; 一剪梅
</yuju>
<![CDATA[
  <00:27.84>真情像草原广阔
  <00:34.06>层层风雨不能阻隔
  <00:41.15>总有云开 日出时候
  <00:48.46>万丈阳光照亮你我
  <00:53.28>
  <00:54.80>真情像梅花开遍
  <01:01.66>冷冷冰雪不能掩没
  <01:08.44>就在最冷 枝头绽放
  <01:15.78>看见春天走向你我
]]>
</geci>
```

(3) 保存修改好的代码。

3.5.5　运行结果

执行程序，显示的效果如图 3-7 所示。

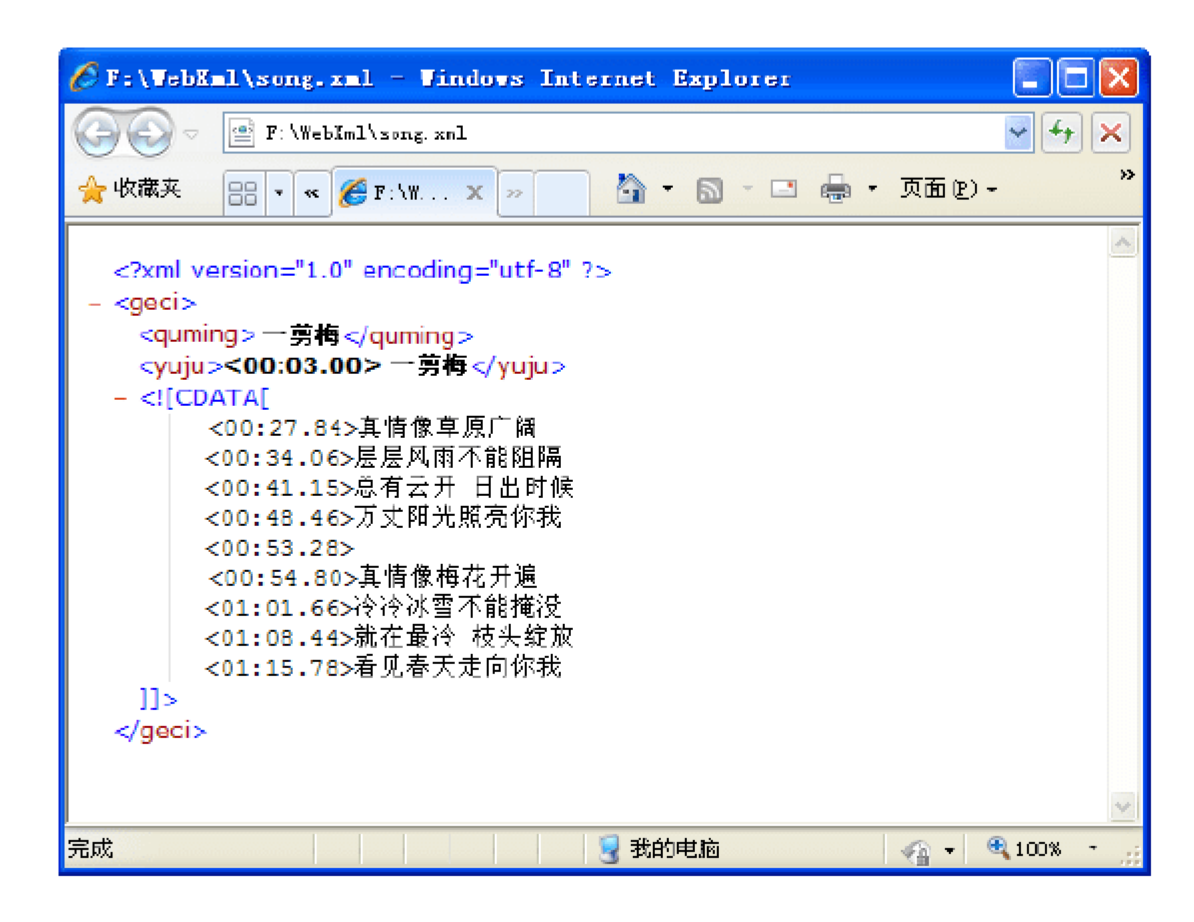


图 3-7　歌词显示的效果

3.5.6　实例分析


源码解析：

在本实例中，我们使用了字符和实体引用，从而在文档中将那些特殊的符号显示出来。另外也使用 CDATA 使歌词的内容不被 XML 解析器解析。

这样看来 CDATA 的使用是不是很简单啊。

3.6　制作精彩的树状后台

当我们打开一张网页，映入眼帘的不仅仅是绚丽多姿的画面，还有丰富多彩的内容。这些内容我们难道要从数据库中一条条地插入进去吗？当然不是，目前无论大小网站或者系统，开发者都会将它们分为前台和后台两部分。前台为客户提供数据库中内容的展示，而后台为该网站或者系统的管理员提供增、删、改、查等功能。

一般后台的展示都是以树状的形式展现的，实现此效果有很多方式，在这里我们使用 DTD 的方式来展示一下。


视频教学：光盘/videos/03/DTD.avi　　　　　　**长度：**27 分钟

3.6.1　基础知识——文档类型定义 DTD

如果想要创建有效的 XML 文档，不仅需要语法正确而且规范，还需要遵循一定的结构规范。而文档类型定义 DTD 恰为 XML 文档的结构规范提供了一组规则约束。下面我们来介绍何为文档类型定义 DTD。

文档类型定义(Document Type Definition，DTD)，是使用文档类型声明来引入 XML 文档中的。它列出了可用在文档中的元素、属性和实体等以及这些内容的相互之间的关系，另外此 DTD 文档中还包含元素的定义规则，元素间关系的定义规则，元素可使用的属性以及可使用的实体或符号规则。

目前，有很多定义好的 DTD 文件可供我们直接使用，这些写好的 DTD 文件已经根据不同的行业和应用建立了通用的元素和标签规则。使用时，不需要我们自己重新创建，只需在它们的基础上加入所需要的新标识即可。当然，我们也可以创建自己的 DTD 文件，以便能够与 XML 文档配合得天衣无缝。

下面我们就来看一下如何创建 DTD 文档。

1．内部 DTD

看到“内部”DTD 和“外部”DTD，你有没有想到在 HTML 文档中的“内部”样式和“外部”样式呢？没错，道理都是一样的。DTD 可以在 XML 文档中直接写入，也可以形成单独的文件，因此将 DTD 分为内部 DTD 和外部 DTD 两种类型。而这两种类型的差别在于：内部定

义好的 DTD 只能在此 XML 文档中使用，外部编辑好的 DTD 则可以被不同的 XML 文档共享和调用。

下面我们先来介绍内部 DTD。

内部 DTD 被称为 DTD 的内部子集，在 XML 文档中直接定义，规则是以“<!DOCTYPE”开始，以“]>”结束，其语法格式如下所示。

```
<!DOCTYPE 根元素[
  <!ELEMENT 根元素(子元素列表)>
  <!ELEMENT 子元素名称 EMPTY/ANY/#PCDTAT/(子元素内容)/(混含内容) >
  <!ATTLIST 元素名 属性名 属性类型 属性默认值>
]>
```

在上述语法中，参数说明如下所示。

- <!DOCTYPE：关键字，表示定义 DTD，且必须为大写。
- <!ELEMENT：定义文档构成的元素。首先，需要声明根元素及其包含的所有子元素。如果文档根元素下有很多子元素，那么就需要多个<!ELEMENT 定义，而 EMPTY / ANY/ #PCDTAT / (子元素内容)/ (混含内容)等为元素的内容类型，每一个子元素对应一种元素的内容类型。
- <!ATTLIST：定义文档中的根元素或者子元素的属性。
-]>：定义结束 DTD。

下面通过一个示例来说明一下内部 DTD 的使用，代码如下所示。

```
<博客 博客 ID="dcy123">
  <文章 文章 ID="dcyandly12">私密日记</文章>
  <时间>2011-2-25</时间>
  <标题>那被逼出来的成熟</标题>
  <内容>
    哭的时候没人哄，我学会了坚强；
    怕的时候没人陪，我学会了勇敢；
    烦的时候没人问，我学会了承受；
    累的时候没人可以依靠，我学会了自立……
    就这样我找到了自己，
  </内容>
</博客>
```

上面的这段代码是没有定义 DTD 的，运行程序的结果如图 3-8 所示。

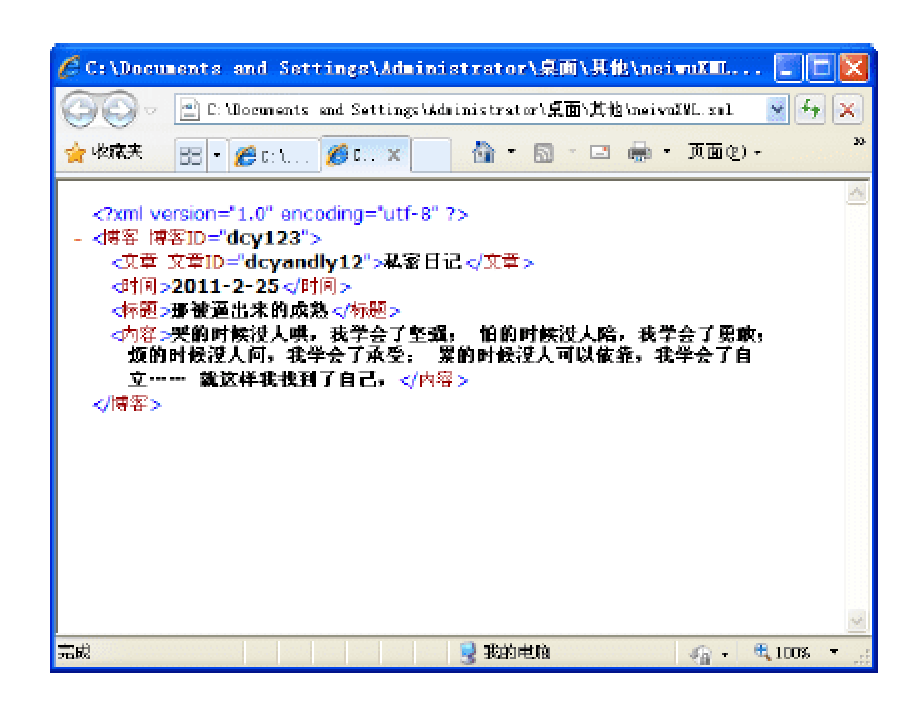


图 3-8　未定义 DTD 的效果

修改代码，在文档内部定义 DTD，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE 博客[
<!ELEMENT 博客 (文章,时间,标题,内容)>
  <!ATTLIST 博客 博客ID ID #REQUIRED>
  <!ELEMENT 文章 (#PCDATA)>
  <!ATTLIST 文章 文章ID ID #REQUIRED>
  <!ELEMENT 时间 (#PCDATA)>
  <!ELEMENT 标题 (#PCDATA)>
  <!ELEMENT 内容 (#PCDATA)>
]>
<博客 博客ID="dcy123">
  <文章 文章ID="dcyandly12">私密日记</文章>
  <时间>2011-2-25</时间>
  <标题>那被逼出来的成熟</标题>
  <内容>
    哭的时候没人哄，我学会了坚强；
    怕的时候没人陪，我学会了勇敢；
    烦的时候没人问，我学会了承受；
    累的时候没人可以依靠，我学会了自立……
    就这样我找到了自己，
  </内容>
</博客>
```

运行程序，效果如图 3-9 所示。

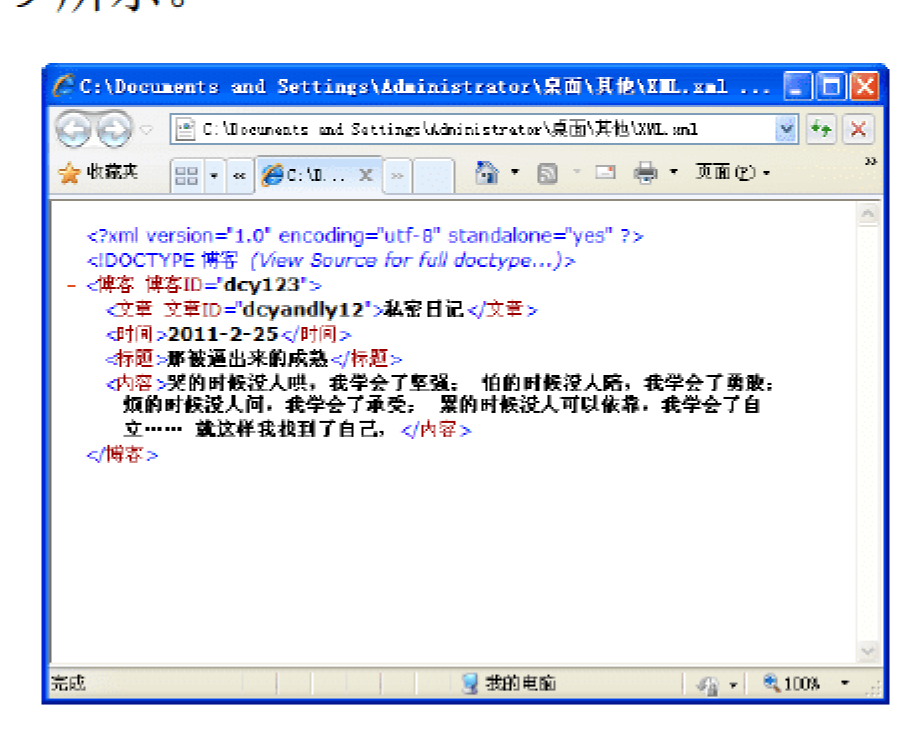


图 3-9　在文档内部定义的 DTD 效果

通过图 3-8 和图 3-9 可以看出，运行结果显示的“<!DOCTYPE 博客 (View Source for full doctype...)>”指定了文档类型的声明。

2．外部 DTD

上面我们介绍了在文档内部定义 DTD，接着来看在外部是如何定义 DTD 的。

所谓外部 DTD，又被称为 DTD 的外部子集，即是指在 XML 文档外单独定义 DTD，并通过外部的 URL 将 DTD 文件引入 XML 文档中。在外部编写的 DTD 文件，扩展名是.dtd。虽然外部 DTD 具有独立性的特征，可以提供给多个 XML 文档使用，但是其定义结构必须与文档中的元素对应，否则没有任何效果。

外部 DTD 与内部 DTD 在 XML 文档中放置的位置相同，有两种语法格式，如下所示：

```
<!DOCTYPE 根元素名 SYSTEM "DTD-URL">
```

或者

```
<!DOCTYPE 根元素名 PUBLIC "DTD-NAME" "DTD-URL">
```

其参数说明如下。

- <!DOCTYPE：关键字，文档类型声明表示包含或引入 DTD。
- SYSTEM：关键字，指的是该外部 DTD 文件是私有的。
- PUBLIC：关键字，指的是该外部 DTD 文件是共有的，而“DTD-NAME”是 DTD 中的一个逻辑名称。
- DTD-URL：通过 URL 将外部 DTD 文件引入 XML 文档中。

我们来看一个小例子。首先，创建一个名称为 waiXMLDTD.dtd 的文件，并添加如下代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT 博客 (文章,时间,标题,内容)>
<!ATTLIST 博客 博客 ID ID #REQUIRED>
<!ELEMENT 文章 (#PCDATA)>
<!ATTLIST 文章 文章 ID ID #REQUIRED>
<!ELEMENT 时间 (#PCDATA)>
<!ELEMENT 标题 (#PCDATA)>
<!ELEMENT 内容 (#PCDATA)>
```

接着，创建一个名称为 waiXML.xml 的文件，并添加如下代码：

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE 博客 SYSTEM "waiXMLDTD.dtd">
<博客 博客 ID="dcy123">
  <文章 文章 ID="dcyandly12">私密日记</文章>
  <时间>2011-2-25</时间>
  <标题>那被逼出来的成熟</标题>
  <内容>哭的时候没人哄，我学会了坚强；
    怕的时候没人陪，我学会了勇敢；
    烦的时候没人问，我学会了承受；
    累的时候没人可以依靠，我学会了自立……
    就这样我找到了自己，</内容>
</博客>
```

保存代码。

运行程序的效果如图 3-10 所示。

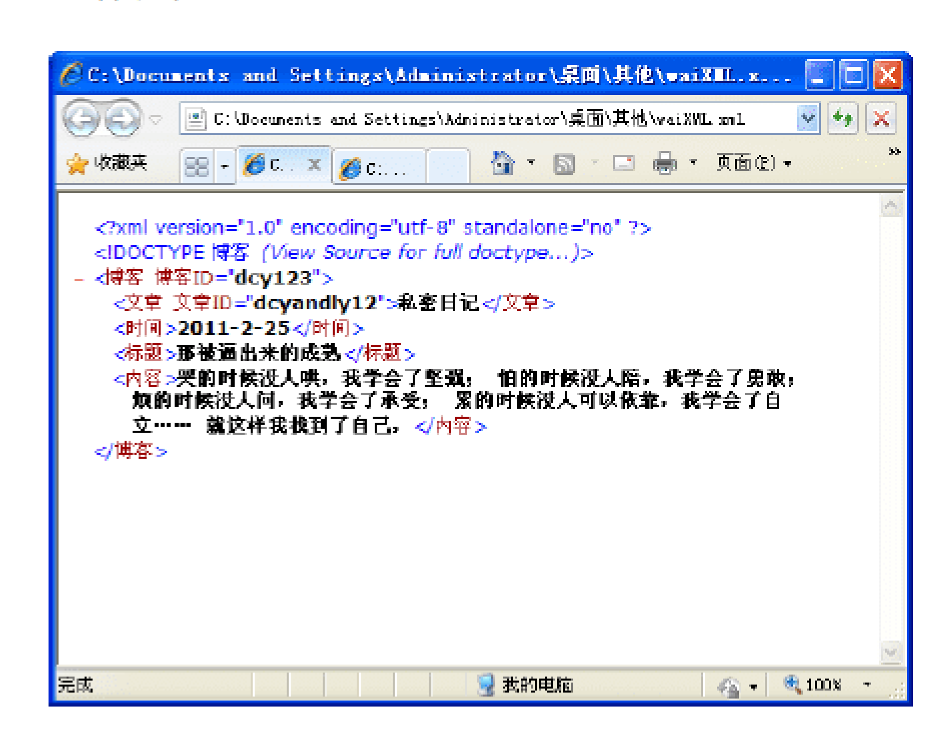


图 3-10　引入外部 DTD 的效果

通过图 3-9 和图 3-10 的对比，可以看出在内部定义 DTD 和使用 URL 引入外部 DTD 的效果是一样的。

3．DTD 对元素和属性的声明

通过上一节的介绍，我们了解了如何在文档内部定义 DTD 和编写外部 DTD 文件，其中 DTD 描述了 XML 文档中元素之间的结构关系。那么这一节就来看看 DTD 中元素和属性的声明。首先来看 DTD 对元素的声明。

元素是 XML 文档中必不可少的构件，它展示了 XML 文档的结构及特性。在 DTD 中使用元素类型来声明 XML 文档中的所有元素。元素类型名称不但可以给出元素的名称，而且还定义了元素的具体类型。元素类型的声明由 ELEMENT 关键字来定义，其语法格式如下所示：

```
<!ELEMENT 元素名称 EMPTY/ANY/#PCDTAT/(子元素内容)/(混含内容)>
```

在此语法中，元素名称后面是元素内容类型。元素内容类型指定了元素要包含的数据内容格式，分别是 EMPTY(空)、ANY(任意)、#PCDATA、子元素型和混合型 5 种类型。下面将对这 5 种类型一一介绍。

(1)　关键字 EMPTY 用于声明空元素。其用法是只能包含属性，而不能包含数据内容或子元素。语法格式如下所示：

```
<!ELEMENT 元素名 EMPTY>
```

例如：在 DTD 中声明元素“瞬间空白”为空元素，代码如下所示。

```
<!ELEMENT 瞬间空白 EMPTY>
```

(2)　ANY 表示该元素可以是 DTD 中定义的其他任何元素或已经编译的字符数据，包括 PCDATA、元素、元素和 PCDATA 的混合内容。其语法格式如下所示：

```
<!ELEMENT 元素名 ANY>
```

例如：在存储图片时，往往因为图片格式的不同而不确定要包含的数据内容，那么使用 ANY 来定义图片的类型是再合适不过了，代码如下所示：

```
<!ELEMENT 图片 ANY>
```

(3)　#PCDATA 表示被解析的字符数据。#PCDATA 类型的元素不能包含任何子元素，只能包含除标记外的字符数据，比如数字、字符和符号等。它的语法格式如下所示：

```
<!ELEMENT 元素名 (#PCDATA)>
```

例如：

```
<!ELEMENT 爱好 (#PCDATA)>
```

(4)　子元素型用于指定某个元素可以包含哪些子元素，语法格式如下所示：

```
<!ELEMENT 元素名 (子元素名称)>
```

例如：

```
<!ELEMENT  图书 (名称，价格)>
```

在此段代码中，表示的是在根元素“图书”下包含两个子元素，分别是“名称”和“价格”。由于这两个子元素是并列的关系，因此用“，”隔开，并且均在 XML 文档中显示，这样的列表形式被称为元素型的序列结构。

除了序列结构，还有一种是选择结构。也就是说，选择定义的根元素的同时也可以选择其根元素要使用的子元素，可选的子元素需要使用“|”隔开。例如：

```
<!ELEMENT 性别(男|女)>
```

在上述代码中，元素“性别”必须从子元素“男”或“女”中选择一个作为子元素，绝对不能包含两个。

(5) 混合型定义的元素不但可以包含子元素，而且可以包含可解析的字符数据，其语法格式如下所示：

```
<!ELEMENT 根元素 (#PCDATA|子元素)*>
```

在此语法中，混合内容元素的声明必须遵循这一格式，即以#PCDATA 开始，后面是混合内容中可能出现的子元素类型。而星号“*”代表元素指示符，需要放在右括号外侧，表示元素可以多次出现。

在 DTD 中，可以作为元素指示符的还有几个，下面我们通过表 3-2 来将这些指示符的作用罗列出来。

表 3-2 元素指示符

元素指示符	作 用
+	元素可以出现任意多次，至少出现一次(≥1)
*	元素可以出现任意多次(≥0)
?	元素要么出现一次要么不出现(0 或 1)

对于我们经常挤公交的人来说，刷卡和投币都不陌生吧。从每位乘客的角度来分析，必须刷卡或者投币方能乘坐公交车。但从众多乘客的角度来分析，刷卡或者投币的情况必定大于等于一次。下面，我们做个小例子。

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE 公交车[
<!ELEMENT 公交车 (乘客,(刷卡,投币+)+ ,上车)>
  <!ELEMENT 乘客 (#PCDATA)>
  <!ELEMENT 刷卡 (#PCDATA)>
  <!ELEMENT 投币 (#PCDATA)>
]>
<公交车>
  <乘客>A 乘客</乘客>
  <刷卡>月票</刷卡>
  <投币>1 元</投币>

  <刷卡>电子钱包</刷卡>
  <投币>1 元</投币>
  <刷卡>月票</刷卡>
```

```
  <投币>1 元</投币>
</公交车>
```

在该文档的 DTD 中，定义了根元素“公交车”及其四个子元素，其中刷卡和投币至少出现一次。

到目前为止，DTD 对元素的声明就告一段落了。下面我们来看一下 DTD 对属性的声明。

属性的作用主要是用来描述元素的额外信息的。在 DTD 中，我们使用 ATTLIST 关键字来定义元素的属性，其语法格式如下所示。

```
<!ATTLIST 元素名 属性名 属性类型 属性默认值>
```

在 DTD 中有以下几种属性类型，如表 3-3 所示。

表 3-3　DTD 中的属性类型

属性类型	含　义
CDATA	字符数据，即没有标记的文本
枚举	可选择的可能值列表
ID	不被文档中任何其他 ID 类型属性共享的唯一名称
IDREF	文档中元素的 ID 类型的属性值
IDREFS	由空格分隔元素的多个 ID
ENTITY	在 DTD 中声明的实体的名称
ENTITIES	在 DTD 中声明由空格分隔的多个实体的名称
NMTOKEN	XML 名称记号
NMTOKENS	由空格分隔的多个 XML 名称记号
NOTATION	在 DTD 中可以向应用程序指定一个外部的处理程序

在 DTD 中的属性默认值如表 3-4 所示。

表 3-4　DTD 中的属性默认值

属性默认值	含　义
#REQUIRED	元素的每个实例必须包含该属性
#IMPLIED	元素实例可以选择是否包含该属性
#FIXED+默认值	属性的值永远定位默认值，如果元素中不包含该属性的属性值，解析器会将默认值作为属性值
默认值	如果元素中不包含该属性的属性值，解析器会将默认值作为属性值，否则该属性可以有其他属性值

4．DTD 中的实体

前面为我们介绍了 5 种预定义的实体引用，分别用来引用 5 个内置的符号实体，只有这些是远远不够的。因此 XML 提供了实体定义，可以使用户将经常使用的文本段定义为实体，这样，用户就能够快速地引用该文本段的实体了。

在 DTD 中，主要分为两种实体，分别是普通实体和参数实体。首先我们来看一下普通实体。

1)　普通实体

普通实体在文档中的格式是：以“&”开始，以“;”结束，在两字符之间的是在 DTD 中定义的实体名称。普通实体又可分为内部普通实体和外部普通实体。内部普通实体是在 DTD 的内部定义，从而被文档引用的实体，语法格式如下所示：

```
<!ENTITY name "text">
```

在此语法中，ENTITY 指的是实体关键字，表明定义的是一个实体。name 指的是实体引用名称，可以在 XML 文档中使用的实体引用。“text”指的是实体的内容。

下面我们来看一个例子，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE 公交车师傅[
  <!ENTITY title "爱情公寓搞笑台词">
  <!ELEMENT 公交车师傅 (标题,展播,保镖) >
  <!ENTITY zhanbo "你，要么刷卡，要么投币，扭什么扭！">
  <!ENTITY baobiao "哎，回来！要么刷卡，要么投币，看什么看！公交车都坐不起，还冒充黑客帝国，哼！">
  <!ELEMENT 标题 (#PCDATA)>
  <!ELEMENT 展播 (#PCDATA)>
  <!ELEMENT 保镖 (#PCDATA)>
]>
<公交车师傅>
  <标题>&title;</标题>
  <展播>&zhanbo;</展播>
  <保镖>&baobiao;</保镖>
</公交车师傅>
```

在该文档中的 DTD 中，我们定义了三个内部普通实体，分别是 title、zhanbo 和 baobiao。然后定义文档中的根元素及其子元素。运行该 XML 文档，文档中引用实体的地方将被实体的内容所代替。执行的结果如图 3-11 所示。

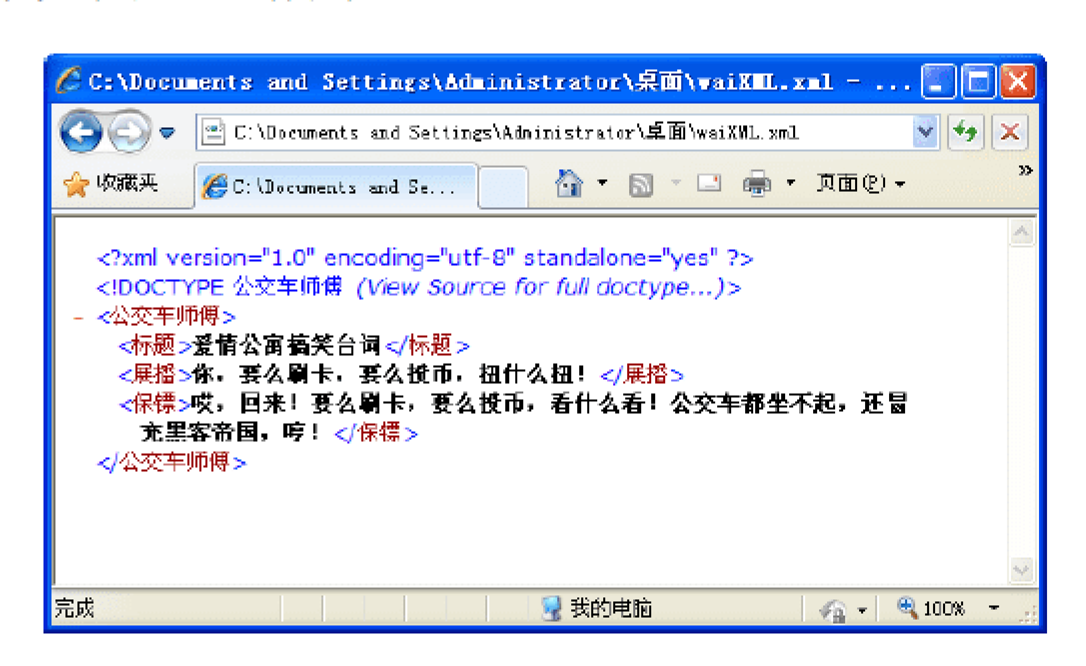


图 3-11　引用内部普通实体的效果

外部普通实体是指实体引用文件在文档外部，通过 URI 引用到 DTD 中，之后方可被文档引用，来看一下它的语法格式。

```
<!ENTITY name SYSTEM "URI">
```

在该语法中，name 为实体的引用名，URI 为有效的链接资源。下面我们使用外部普通实体来做一个小例子，以便加深我们对外部实体的引用，如下所示：

首先，创建一个名称为 XML.xml 的文件，并添加如下代码：

```
<?xml version="1.0" encoding="utf-8" ?>
哎，回来！要么刷卡，要么投币，看什么看！公交车都坐不起，还冒充黑客帝国，哼！
```

然后再创建一个名称为 wai.xml 的文件，代码如下：

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE 公交车师傅[
  <!ENTITY title "爱情公寓搞笑台词">
  <!ELEMENT 公交车师傅 (标题,保镖) >
  <!ENTITY zhanbo "你，要么刷卡，要么投币，扭什么扭！">
  <!ENTITY baobiao SYSTEM "XML.xml">
  <!ELEMENT 标题 (#PCDATA)>
  <!ELEMENT 保镖 (#PCDATA)>
]>
<公交车师傅>
  <标题>&title;</标题>
  <保镖>&baobiao;</保镖>
</公交车师傅>
```

保存代码。

运行程序，执行的结果如图 3-12 所示。

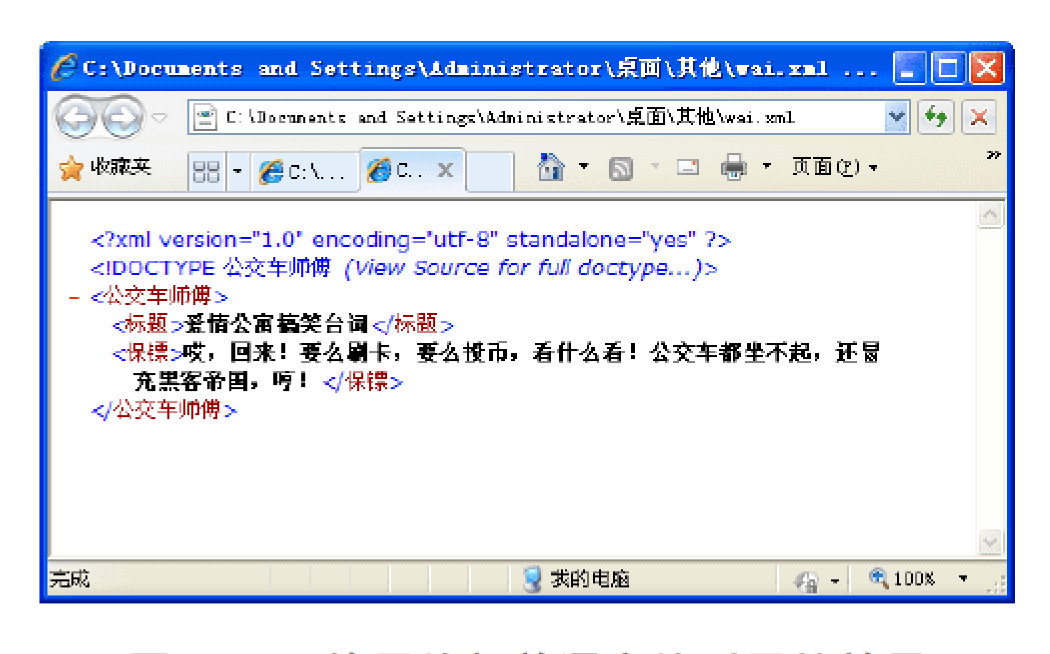


图 3-12　使用外部普通实体引用的效果

通过上面的介绍，我们了解了什么是普通实体，以及内部和外部普通实体的使用。接下来要讲的是参数实体。

2)　参数实体

普通实体在 DTD 中定义，在 XML 文档中引用，通过解析器将实体变成文档中的一部分。而参数实体只能在 DTD 中定义，并且在 DTD 中使用，而与 XML 文档无关。

参数实体引用是以“%”开头，以“;”结束的。由于参数实体引用的方式不同，因此可以划分为内部参数实体引用和外部参数实体引用。

声明内部参数实体的格式如下所示：

```
<!ENTITY % name "text">
```

在该语法中，name 指的是参数实体的引用名称，text 指的是参数实体的内容。

下面我们通过一个小例子来说明一下。

首先创建一个名称为 neiCan.dtd 文件，并添加如下代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT 班级 (Y20711) >
<!ENTITY % name "姓名" >
<!ENTITY Y20711 "%name;" >
<!ELEMENT 姓名 (#PCDATA) >
<!ELEMENT 成绩 (#PCDATA) >
<!ELEMENT Y20711 (#PCDATA)>
```

在上述代码中，声明了内部参数实体 name，接着在内部使用“%name;”的方式引用。

接着，创建一个名称为 neiCan.xml 的文件，添加如下代码：

```
<?xml version="1.0" encoding="utf-8"  standalone="no"?>
<!DOCTYPE 班级 SYSTEM  "neiCan.dtd" >
<班级>
  <Y20711>
    <姓名>dcy</姓名>
  </Y20711>
</班级>
```

保存完成的代码。

运行程序的结果如图 3-13 所示。

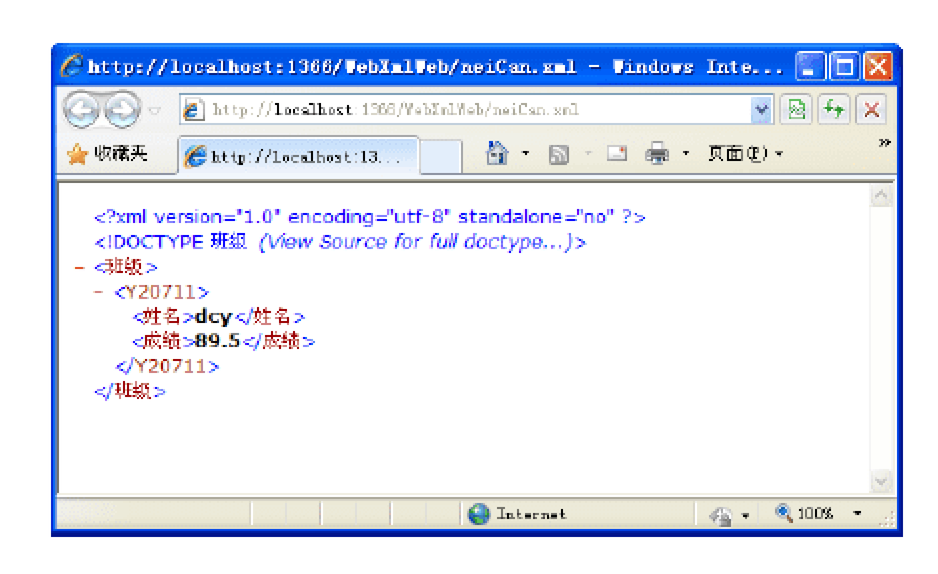


图 3-13　内部参数实体引用的效果

接着我们来看外部参数实体的引用。

在 XML 文档中，会有很多的属性和元素，那么 DTD 的声明就会比较复杂，因此不利于以后的维护和更新。有时，用户有可能在多个文档中需要声明 DTD 文件中同一部分的元素和属性，这样就会造成代码的冗余。而外部参数实体就可以解决这个问题，来增强元素声明的通用性。

外部参数实体是指一个 DTD 可以链接到另一个 DTD 上，从而得到第一个 DTD 中的元素以及属性的声明。其语法格式如下所示。

```
<!ENTITY % name "URI">
```

在该语法中，name 为外部参数实体的引用名称，URI 为外部 DTD 的 URL。

下面我们来看一个例子。

首先，创建一个名为 canDTD.dtd 的文件，其代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT 学生 (姓名,成绩) >
<!ELEMENT 姓名 (#PCDATA)>
```

```
<!ELEMENT 成绩 (#PCDATA)>
```

然后，新添加一个名为 tianCAN.dtd 的文件，其代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT 班级 (班级名称,学生*)>
<!ELEMENT 班级名称 (#PCDATA)>
<!ELEMENT 学生 (#PCDATA)>
<!ENTITY % work SYSTEM "canDTD.dtd">
```

在上述代码中，我们使用了“<!ENTITY % name “URI”>”的方式来引用外部的 DTD 文件。接着，创建一个名称为 waiCAN.xml 的文件，代码如下所示：

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE 班级 SYSTEM "tianCAN.dtd" >
<班级>
  <班级名称>Y20828</班级名称>
<学生>
    <姓名>dcy</姓名>
    <成绩>89</成绩>
</学生>
  <学生>
    <姓名>ly</姓名>
    <成绩>90</成绩>
  </学生>
  <学生>
    <姓名>ldx</姓名>
    <成绩>50</成绩>
  </学生>
  <学生>
    <姓名>lws</姓名>
    <成绩>85</成绩>
  </学生>
</班级>
```

在此段代码中，我们使用“<!DOCTYPE 班级 SYSTEM "tianCAN.dtd">”引用外部参数实体。

运行程序的结果如图 3-14 所示。



图 3-14　引用外部参数实体的效果

3.6.2 实例描述

我有一个网站后台，想用 XML 文档的形式展现出来，刚好在上面我们讲过 DTD 的实体，我有一种想试一试的冲动。

下面就来看一下我的实现方案。

3.6.3 实例应用

【例 3-4】 制作精彩的树状后台

(1) 创建一个 DTD 文件，名称为 ShuZhuang.dtd。

(2) 在 ShuZhuang.dtd 文件中，添加如下代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT 关于我们 (公司简介,荣誉资质,分类管理,子类管理)>
<!ELEMENT 公司简介 (#PCDATA)>
<!ELEMENT 荣誉资质 (#PCDATA)>
<!ELEMENT 分类管理 (#PCDATA)>
<!ELEMENT 子类管理 (#PCDATA)>
<!ELEMENT 新闻中心 (公司新闻,分类管理,子类管理)>
<!ELEMENT 公司新闻 (#PCDATA)>
<!ELEMENT 产品中心 (产品展示,最新产品,分类管理,子类管理)>
<!ELEMENT 产品展示 (#PCDATA)>
<!ELEMENT 最新产品 (#PCDATA)>
<!ELEMENT 客户服务 (客户服务 1,分类管理,子类管理)>
<!ELEMENT 客户服务 1 (#PCDATA)>
<!ELEMENT 经典案例 (分类管理,子类管理)>
```

(3) 创建一个名称为 ShuZhuangTian.dtd 文件。

(4) 在 ShuZhuangTian.dtd 文件中添加如下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT 后台管理 (后台名称,关于我们,新闻中心,产品中心,客户服务,经典案例*)*>
<!ELEMENT 后台名称 (#PCDATA)>
<!ELEMENT 关于我们 (#PCDATA)>
<!ELEMENT 新闻中心 (#PCDATA)>
<!ELEMENT 产品中心 (#PCDATA)>
<!ELEMENT 客户服务 (#PCDATA)>
<!ELEMENT 经典案例 (#PCDATA)>
<!ENTITY % work SYSTEM "ShuZhuang.dtd">
```

(5) 创建一个名称为 ShuZhuang.xml 文件，并添加如下代码：

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE 后台管理 SYSTEM "ShuZhuangTian.dtd">
<后台管理>
  <后台名称>我的后台管理</后台名称>
  <关于我们>
    <公司简介></公司简介>
```

```
    <荣誉资质></荣誉资质>
    <分类管理></分类管理>
    <子类管理></子类管理>
  </关于我们>
  <新闻中心>
    <公司新闻></公司新闻>
  </新闻中心>
  <产品中心>
    <产品展示></产品展示>
    <最新产品></最新产品>
  </产品中心>
  <客户服务>
    <客户服务 1></客户服务 1>
  </客户服务>
  <经典案例>
    <分类管理></分类管理>
    <子类管理></子类管理>
  </经典案例>
</后台管理>
```

保存修改好的代码。

3.6.4　运行结果

运行程序的结果如图 3-15 所示。

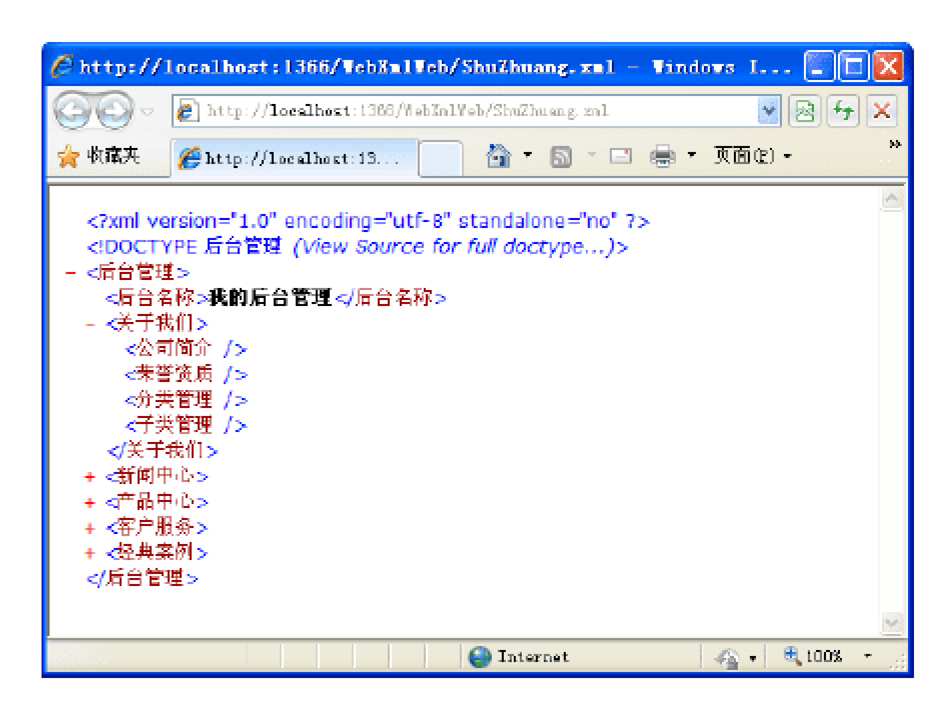
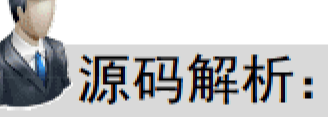


图 3-15　XML 的后台展示效果

3.6.5　实例分析



源码解析：

在本实例中，ShuZhuang.xml 中引用了外部参数实体 ShuZhuangTian.dtd 文件。ShuZhuangTian.dtd 文件又链接了 ShuZhuang.dtd 文件，从而完成了此网站后台的目录结构。

是不是很简单啊。

3.7 常见问题解答

3.7.1 关于 XML 中命名空间的问题



关于 XML 中命名空间的问题。
网络课堂：http://bbs.itzcn.com/thread-15024-1-1.html

创建了一个 XML 文件，其中主要包括两部分，第一部分使用 table 元素包含了水果的信息。代码如下所示：

```
<table>
      <tr>
         <td>Apples</td>
         <td>Bananas</td>
      </tr>
</table>
```

第二部分又使用 table 元素包含了桌子的信息，如下所示：

```
<table>
        <name>African Coffee Table</name>
        <width>80</width>
        <length>120</length>
</table>
```

由于这两个 XML 代码都包含了 table 元素，但是这两个 table 元素的定义和所包含的内容各不相同，我该怎么办才能将这两个 XML 代码中的 table 元素给区别开呢？

【解决办法】

在 XML 文档中，你可以使用命名空间将这两个 XML 代码中的 table 元素区别开。

第一部分包含水果的信息的 table 元素，代码如下所示：

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

第二部分包含桌子的信息的 table 元素，代码如下所示：

```
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

在上面的例子中除了使用前缀外，两个 table 元素都使用了 xmlns 属性，使元素和不同的命名空间关联到一起。

3.7.2　XML 中的 CDATA 区和注释有什么区别



XML 中的 CDATA 区和注释有什么区别？
网络课堂：http://bbs.itzcn.com/thread-15025-1-1.html

最近我正在学习 XML，但学到 CDATA 区时，我甚感迷惑。因为我觉得 CDATA 和注释的功能是一样的，都是不被解析器解析。希望能具体分析一下，谢谢大侠。

【解决办法】

首先，CDATA 的功能是还原该语句的本来含义。在 XML 文档中“<”这个符号是一个节点的开始符号。例如：有一个字符串“我家门前有棵葡萄树”，如果将该字符串放在节点中间，当解析器解析到“<”的时候就会报错。但是，当我们加上 CDATA 的时候，意思就是告诉解析器，这里面的内容不希望被解析，而是希望直接输出。

3.7.3　XML 文件引用外部 DTD 文件的问题



XML 文件引用外部 DTD 文件的问题。
网络课堂：http://bbs.itzcn.com/thread-15026-1-1.html

我创建了一个 XML 文件，其代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE Title SYSTEM "xy1.dtd">
<Title>[ 部 2010-7-14 11:08:47]</Title>
```

在该代码中引用了 DTD 文件，代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!ENTITY amp "&">
<!ENTITY reg "®">
<!ENTITY nbsp " ">
```

这两个文件在同一路径下，但运行时提示“XML 解析错误：未定义的实体”。

```
位置：http://xiaobin.com/xy.xml
行：9，列：9：
<Title>[ 专线部 2010-7-14 11:08:47]</Title>，在线等……
```

【解决办法】

你只需要简单地将 DTD 文件修改一下即可，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT Title (#PCDATA)>
<!ENTITY reg "®">
<!ENTITY nbsp " ">
```

保存之后再运行，效果如图 3-16 所示。

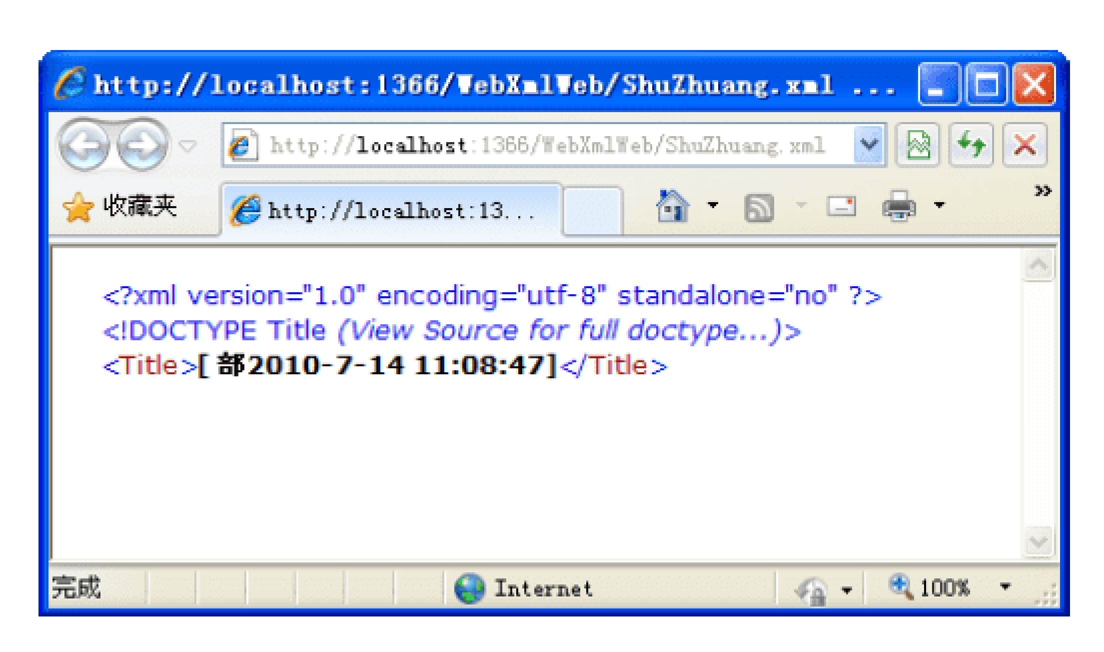


图 3-16　成功运行时的效果

3.8　习　　题

一、填空题

(1) 在 XML 文档中，__________表示使用 DTD 的形式，值只能是 yes 或者 no。

(2) 在 XML 文档中声明命名空间，使用的关键字是__________。

(3) XML 文档中的字符“<”和“&gt”，其中“<”字符在文档中显示的效果是“<”，那么字符“&gt”在文档中显示的效果是__________。

(4) __________是用来包含文本的方法，通常用于建立代码的脚本，如 JavaScript、VBScript 等。

(5) 声明文档类型定义 DTD，需要使用关键字__________。

(6) 在 DTD 对元素的声明过程中，其中__________用来表示空元素。

(7) XML 提供了实体定义，在 DTD 中主要分为普通实体和__________。

二、选择题

(1) 在 XML 文档的序言中包括 XML 的声明和注释两部分，其中在 XML 的声明中，standalone 表示是否引用外部实体，当 standalone 的值为__________时，说明需要引用外部实体。

A. yes　　B. no　　C. ture　　D. false

(2) 在下面的几个选项中，__________声明命名空间是正确的。

A.

```
<goods xmlns:info="http://www.qxian.com/SkinCare">
```

B.

```
<goods info="http://www.qxian.com/SkinCare">
```

C.

```
<goods xmlns:info:http://www.qxian.com/SkinCare">
```

D.

```
<goods xmlns:info http://www.qxian.com/SkinCare">
```

(3) 有这样一个作者，他出了很多本图书，并且有的一本图书还可以被不同的出版社出版，且作者简历可有可无。那么在 DTD 中声明该图书元素正确的是__________。

A.

```
<!DOCTYPE 图书[
<!ELEMENT 图书 (作者,(书名,出版社+)+ ,作者简历?)>
  <!ELEMENT 作者 (#PCDATA)>
  <!ELEMENT 书名 (#PCDATA)>
  <!ELEMENT 出版社 (#PCDATA)>
]>
```

B.

```
<!DOCTYPE 图书[
<!ELEMENT 图书 (作者,(书名,出版社+)+ ,作者简历)>
  <!ELEMENT 作者 (#PCDATA)>
  <!ELEMENT 书名 (#PCDATA)>
  <!ELEMENT 出版社 (#PCDATA)>
  <!ELEMENT 作者简历 (#PCDATA)>
]>
```

C.

```
<!DOCTYPE 图书[
<!ELEMENT 图书 (作者,(书名,出版社+)+ ,作者简历?)>
  <!ELEMENT 作者 (#PCDATA)>
  <!ELEMENT 书名 (#PCDATA)>
  <!ELEMENT 出版社 (#PCDATA)>
  <!ELEMENT 作者简历 (#PCDATA)>
]>
```

D.

```
<!DOCTYPE 图书[
<!ELEMENT 图书 (作者,(书名,出版社+)+ ,作者简历?)>
  <!ELEMENT 作者 (#PCDATA)>
  <!ELEMENT 出版社 (#PCDATA)>
  <!ELEMENT 作者简历 (#PCDATA)>
]>
```

(4) 类似于一个单选按钮，例如：男或女，需要选择其中的一个元素，使其显示在 XML 文档上，那么我们可以使用__________。

A.

```
<!ELEMENT 性别(男#女)>
```

B.

```
<!ELEMENT 性别(男|,女)>
```

C.

```
<!ELEMENT 性别(男，女)>
```

D.

```
<!ELEMENT 性别(男|女)>
```

三、上机练习

上机练习 1： 内部普通实体的引用。

通过对本章的学习，我们对 XML 的基础有了大致的了解。例如：如何声明 XML、XML 的标记与元素、在 XML 中字符和实体的引用以及 CDATA 的使用，另外本章还介绍了文档类型定义 DTD。那么这次的上机练习，针对的是实体引用。

在本次上机练习中，我们使用的是内部引用普通实体，下面将给出实体引用的代码：

```
<诗句>
  <标题>&titile;</标题>
  <ldy>&ldy;</ldy>
</诗句>
```

使其显示效果如图 3-17 所示。

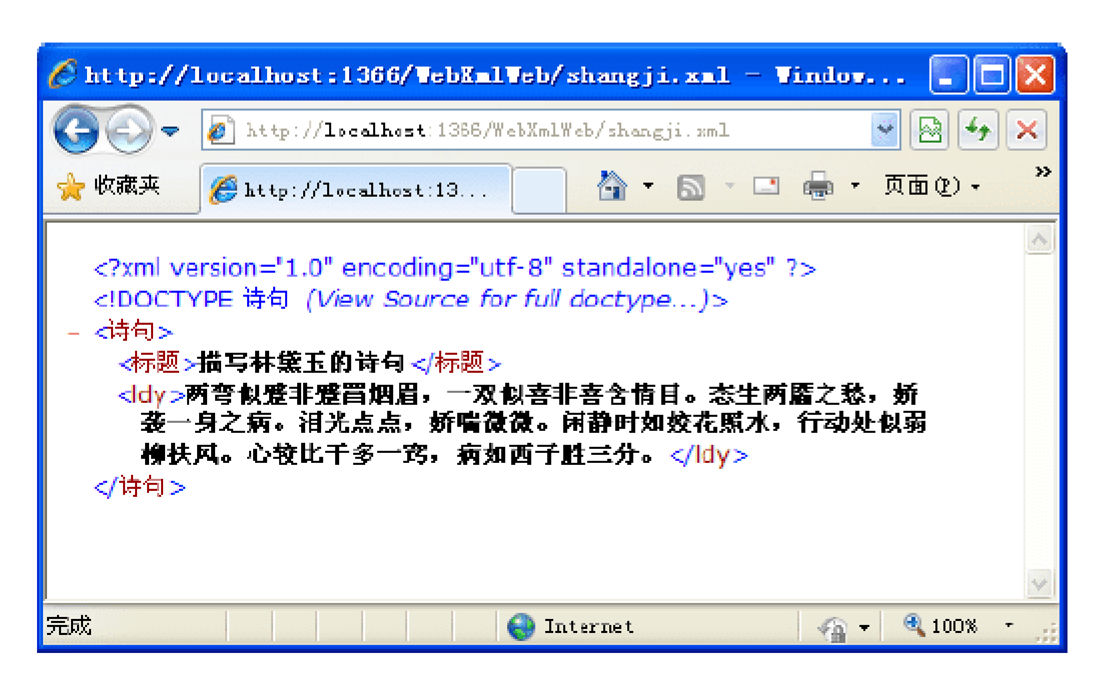


图 3-17　最终的显示效果



第 4 章　Web 服务类型系统——XSD

内容摘要：

众所周知，Web 服务的优势之一就是能够跨平台、跨系统。那么在这些平台上调用时如何正确地处理和传递数据便成为困扰开发人员的一个问题。例如，一些语言使用 32 位来存储整数，而其他语言只使用 16 位来存储整数等类似情况。当试图在这些不同的系统之间进行通信时，就会导致彼此根本无法理解与转换，甚至使系统崩溃。

好在 Web 服务的另一个优势是基于 XML 的。这就意味着用来传递 Web 服务数据的 XML 不应该存在特定于系统或平台的数据类型。也就是说，需要在 XML 中使用一个各种平台都支持、都理解的类型系统来传递类型信息。

而目前，属于 XML 规范的 XML Schema Definition 所定义的类型系统已经被很多软件开发商支持。因此，Web 服务为了解决类型不兼容问题理所当然地采用这个类型系统，即 XSD 类型系统。

学习目标：

- 了解 XSD 与 DTD 的区别
- 掌握 XSD 文档的创建方式和命名空间的使用
- 掌握如何为简易元素定义数据类型
- 掌握在 XSD 中对元素进行限制、联合和列表的方法
- 熟悉匿名类型的使用
- 掌握如何定义一个复杂数据类型
- 掌握简单类型和纯元素类型的使用方法
- 了解混合类型和空类型的使用方法
- 掌握如何声明 XSD 元素和属性
- 熟悉如何通过程序验证 XSD 文档
- 了解二进制数据的传输方法

4.1 什么是 XSD

XSD 是 XML Schema Definition 的简写，也被称为 XML Schema 语言，它是以 XML 语言为基础的。同 DTD 一样，XSD 也是描述 XML 文档结构的一种方式。但是，除了描述文档结构外，XSD 还能限制文本数据的实际类型，这是 DTD 所不能比拟的。

4.1.1 网络教学

视频教学：光盘/videos/04/什么是 XSD.avi　　**长度：**5 分钟

4.1.2 基础知识——XSD 简介

上一章我们学习了 DTD，使用它能够约束 XML 文件的标记和树形结构，而不涉及文本的具体内容。但如果需要指定标记内容的“数据类型”(例如整数、小数、字符数据等)，那么 XSD 则是最佳的选择。

可以说 XSD 是 DTD 的一个替代产品，XSD 由一套预先定义的 XML 元素和属性组成，具有一致性、扩展性、互换性、规范性、数据类型多样性等特点。使用 XSD 可以完成如下工作：

- 定义可出现在文档中的元素；
- 定义可出现在文档中的属性；
- 定义哪个元素是子元素；
- 定义子元素的次序；
- 定义子元素的数目；
- 定义元素是否为空，或者是否可包含文本；
- 定义元素和属性的数据类型；
- 定义元素和属性的默认值以及固定值。

我们可以用一个指定的 XSD 来验证某个 XML 文档，以检查该 XML 文档是否符合其要求。还可以通过 XSD 指定一个 XML 文档所允许的结构和内容，并可据此检查一个 XML 文档是否是有效的。

另外，由于 XSD 本身是一个 XML 文档，因此可以用通用的 XML 解析器解析它。

目前 XSD 有两种标准：Microsoft XML Schema 和 W3C XML Schema。这两种标准在定义 XML 文档时是相同的，只是结构声明、描述、命名空间的使用和 Schema 文件的后缀名不同。在这里，我们以最常用的 W3C XML Schema 为例进行介绍。

在 W3C XML Schema 标准中，根元素必须以“<xs:schema>”或“<xsd:schema>”开始，而且所有元素及属性的声明均是以“xsd”和“xs”开头。XSD 文件以“.xsd”为扩展名进行保存。

例如，下面给出一个 XSD 文档的结构代码：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="命名空间">
  元素声明部分或属性声明部分
</xs:schema>
```

通常引用的命名空间为“http://www.w3.org/2001/XMLSchema”，元素或属性的声明视文档内容而定。

例如，要发送一条短信，首先必须有收信人和发信人，此外标题以及信息内容也是必不可少的。如下的代码演示了使用 XSD 来定义短信 XML 文档中的元素结构和类型：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="message">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="to" type="xs:string"/>
            <xs:element name="from" type="xs:string"/>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

这只是一个简单示例，旨在帮助读者快速了解 XSD 的概况。在本节后面将详细学习各个部分的内容。

4.2 根据 XML 文件定义简易元素

在开发和使用 Web 服务的过程中，经常需要创建新的 XSD 架构或者读取并理解已有的 XSD 架构，甚至有时候需要修改 XSD 架构。在这一节中，我们来学习基于 XSD 类型系统如何定义最基本的元素。

 **视频教学：**光盘/videos/04/定义简易元素.avi 长度：9 分钟

4.2.1 基础知识——定义简易元素

简易元素具有不可再分性，是指仅包含文本的元素。它不会再包含其他任何元素或属性。不过，“仅包含文本”可不是你想象中的那样，文本其实具有很多类型，它可以是内置数据类型，也可以是自定义的类型。

声明简易元素的语法如下：

```
<xs:element name="名称" type="数据类型"/>
```

此处，name 指元素的名称，type 指元素的类型。XSD 拥有很多内置数据类型，表 4-1 就向我们展示了常用的几个内置数据类型。

表 4-1　常用的内置数据类型

数据类型	说　明
string	字符串型
decimal	十进制数型，包含任意精度和位数
int	有正负的 32 位整数
integer	整型
float	标准的 32 位浮点数，如 11.87
byte	有正负的 8 位整数
boolean	布尔型，元素只能取 true、false、1(表示 true)或者 0(表示 false)
date	日期型，格式为 YYYY-MM-DD
month	日期型，格式为 YYYY-MM
year	年份日期型，格式为 YYYY
time	时间型，格式为 HH:MM:SS
datetime	日期时间型，其形式为 YYYY-MM-DD hh:mm:ss
anyURI	元素包含一个 URL

假设要创建一个用于描述个人档案信息的 XML 文档，应该包括姓名、出生日期、性别、身高、体重、学历和爱好等。如下所示为定义好的 XML 元素：

```
<姓名>祝红涛</姓名>
<出生日期>1990-10-10</出生日期>
<性别>1</性别>
<婚否>false</婚否>
<身高>1.75</身高>
<体重>65</体重>
<星座>狮子座</星座>
<血型>B</血型>
<学历>本科</学历>
<爱好>篮球、唱歌、阅读和旅行</爱好>
<网站>http://www.itzcn.com</网站>
```

可以看到，在这个 XML 文档中除了普通的文本型内容外，还有日期型、数字型、布尔型以及 URL 类型内容。

针对这个 XML 文档，我们可以使用 XSD 的简易元素来定义档案信息中各个选项的数据类型。例如，出生日期必须为日期，而且格式必须为“年-月-日”；性别只能为“0”或者“1”；等等。

如下所示为最终定义好的 XSD 简易元素：

```
<xs:element name="姓名" type="xs:string"/>
<xs:element name="出生日期" type="xs:date"/>
<xs:element name="性别"  type="xs: boolean "/>
<xs:element name="婚否" type="xs:boolean"/>
<xs:element name="身高" type="xs:decimal"/>
<xs:element name="体重" type="xs:integer"/>
```

```
<xs:element name="星座" type="xs:string"/>
<xs:element name="血型" type="xs:string"/>
<xs:element name="学历" type="xs:string"/>
<xs:element name="爱好" type="xs:string"/>
<xs:element name="网站" type="xs:anyURI"/>
```

另外，简易元素还可以有默认值和固定值。当元素没有被指定其他的值时，默认值将被自动赋给元素。如下例所示：

```
<xs:element name="性别"  type="xs: boolean " default="1"/>
```

我们用“1”和“0”来分别表示性别“男”和“女”，默认值“1”表示如果没有给元素“性别”赋值，默认的值为“1”。

固定值同样是自动赋给元素的，同时固定值被赋予后将无法给元素赋其他值。如下例所示：

```
<xs:element name="性别"  type="xs: boolean " fixed="1"/>
```

性别被赋予固定值“1”，表示性别的值将会是表示男的“1”，且无法改变。

4.2.2　实例描述

在本节中，我们学习了如何定义简易元素，以及如何根据 XML 文件来创建 XSD 文件。现在，我们通过一个实例来加深对简易元素的理解。在本实例中，我们将为简易元素应用更多的内置数据类型。

4.2.3　实例应用

【例 4-1】 根据 XML 文件定义简易元素

(1) 新建一个 XML 文件，其代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<information >
   <name>Harry</name>
   <age>10</age>
   <sex>1</sex>
   <like>看书</like>
   <birthday>2000-01-01</birthday>
   <timeout>10:10:00</timeout>
   <ismarry></ismarry>
</information>
```

(2) 将文档保存为 XML 文件格式，名称为“4-例 1.xml”。

(3) 接下来就是创建 XSD 文件。在 VS 2010 中打开该文件，再执行 XML|【创建架构】命令可以自动生成 XSD 文件。当然也可以手动创建，最终的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
    <xs:element name="information">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string" />
                <xs:element name="age" type="xs:unsignedByte" />
                <xs:element name="sex" type="xs:unsignedByte" />
                <xs:element name="like" type="xs:string" />
                <xs:element name="birthday" type="xs:date" />
                <xs:element name="timeout" type="xs:time" />
                <xs:element name="ismarry" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(4) 可以看到，自动生成的 XSD 文件中为简易元素指定的数据类型与实际所期望的有一定差异。因此，这一步在此基础上进行修改。如下所示为修改后各个 XSD 元素的类型：

```
<xs:element name="name" type="xs:string" />
<xs:element name="age" type="xs:byte" />
<xs:element name="sex" type="xs:boolean" />
<xs:element name="like" type="xs:string" />
<xs:element name="birthday" type="xs:date" />
<xs:element name="timeout" type="xs:time" />
<xs:element name="ismarry" type="xs:boolean" default="false" />
```

(5) 现在，XML 文件以及使用的 XSD 架构文件都创建完成了。但是，两者并没有建立任何关联。如果在 XML 文件要引用 XSD 架构，还需要添加如下代码：

```
<information xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="4-例 1.xsd">
```

4.2.4 运行结果

将 XML 文件和 XSD 文件在浏览器中打开，如图 4-1 和图 4-2 所示。

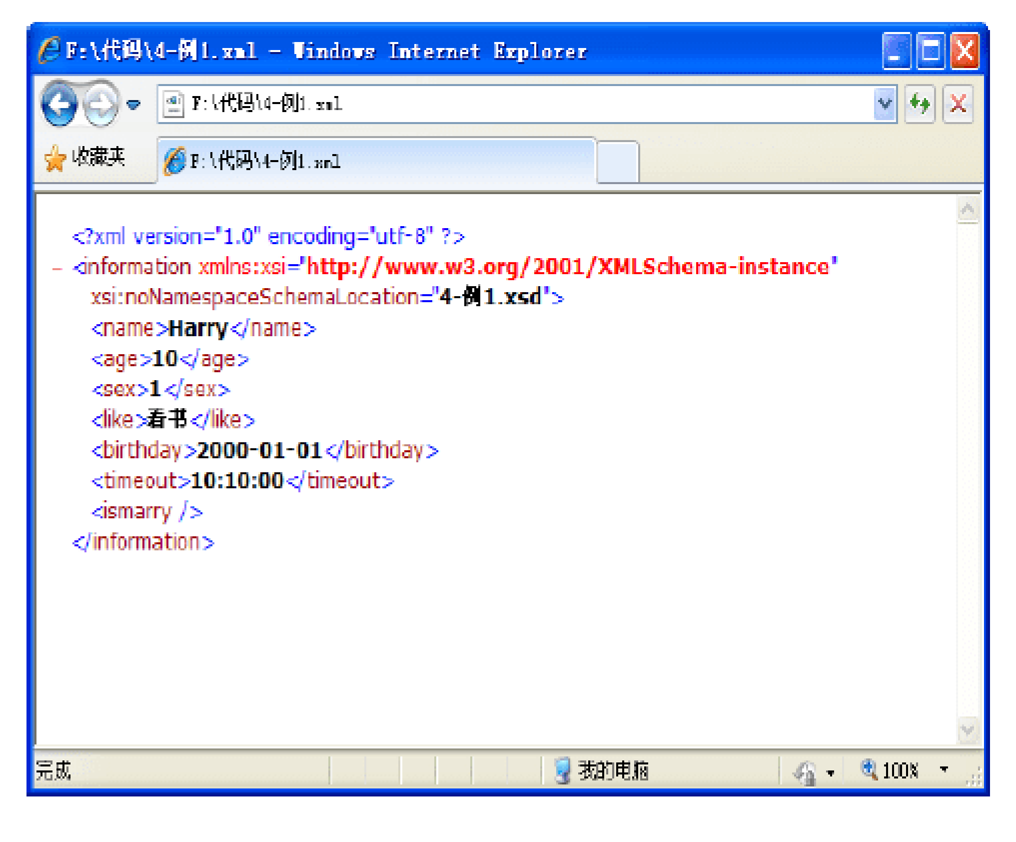


图 4-1 XML 文档

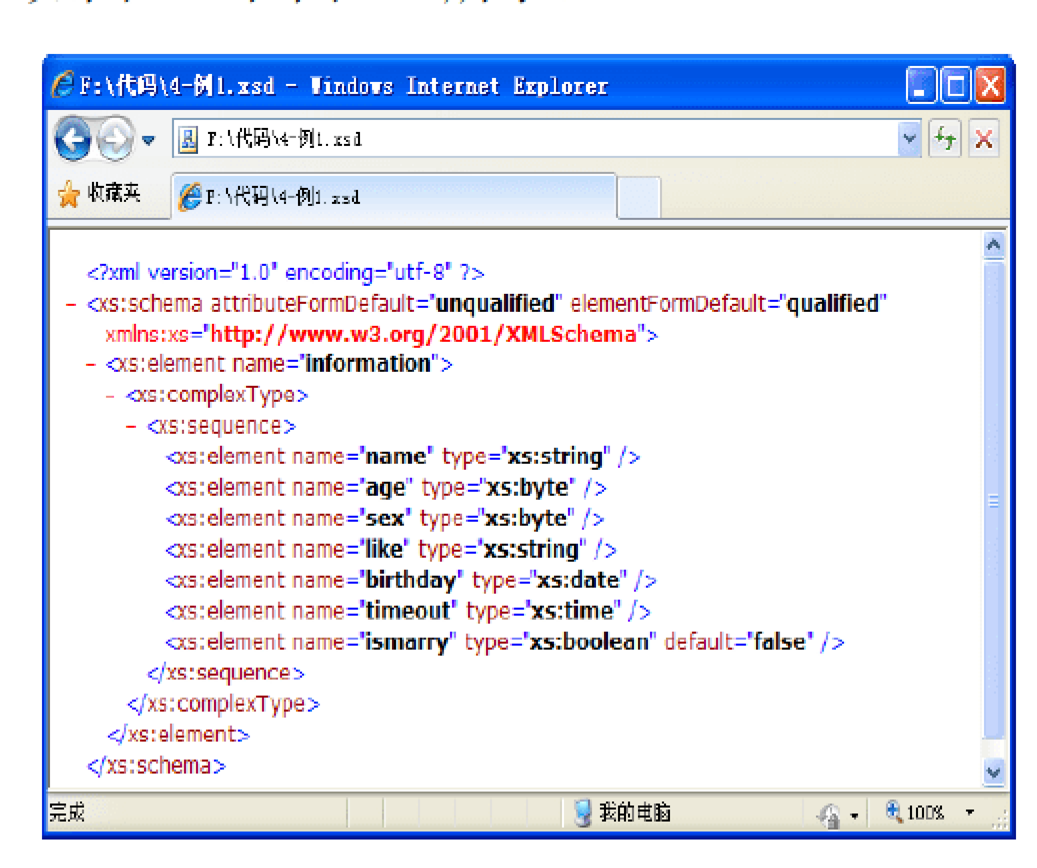


图 4-2 XSD 文档

4.2.5　实例分析



源码解析：

为了表示一个文件为 XSD 架构文件，必须以“.xsd”为扩展名进行保存。还需要指定 http://www.w3.org/2001/XMLSchema 作为命名空间，并添加一个别名引用，一般使用“xs”或者“xsd”。

本实例中的 XML 文档比较简单，没有包含子元素或者属性。因此在 XSD 架构中使用简易元素就可以完成，但是要注意适当安排元素的数据类型。

最后，在 XML 文档中使用 xsi:noNamespaceSchemaLocation 引用 XSD 架构文件时要注意它们的物理保存路径在同一目录下。

4.3　实现图书分类的 XSD

除了上节介绍的 XSD 内置数据类型外，XSD 还允许开发人员自定义与应用程序相关的数据类型，因而扩展了 XSD 的数据类型库，并增强了 XSD 应用的灵活性。

根据数据类型的定义方式，W3C XSD 主要可分为两种：简单类型和复杂类型。本节我们先来学习简单数据类型。

**视频教学：**光盘/videos/04/实现图书分类.avi　　　　长度：17 分钟

4.3.1　基础知识——定义简单数据类型

通常情况下，XML 中的元素会包含属性，有时还会包含纯文本或者子元素。如果一个元素仅仅包含数字、字符串或其他文本数据，但不包括子元素和属性，那么这种元素被称为简单类型。

例如，下面通过描述手机信息的 XML 文档形象地说明 XSD 中简单类型与复杂类型的区别。

```
<手机>
<!-- 这是一个包含属性的复杂类型的元素 -->
<型号 品牌="NOKIA">1100</型号>
<!-- 这是一个包含子元素的复杂类型的元素 -->
<主要参数>
<体积>113×43.5×15.5mm </体积>
<通话时间>408 分钟</通话时间>
<屏幕参数>26 万色 TFT 彩色；240×320 像素；2.0 英寸</屏幕参数>
</主要参数>
<!-- 这是一个简单类型的元素 -->
```

```
<网络频率>GSM/GPRS/EDGE</网络频率>
</手机>
```

简而言之，简单数据类型就是在 XSD 内置数据类型基础上，通过限制基类型、列表或者联合的方式定义的新数据类型。

XSD 简单类型由 simpleType 关键字声明，其语法格式如下：

```
<simpleType name =elementName  id = elementId >
  <annotation|restriction | list | union>…
</simpleType>
```

语法中各参数的定义如下。

- name：类型名称，该名称必须是在 XML 命名空间规范中定义的无冒号名称。
- id：该元素的 ID。id 值必须属于类型 ID 并且在包含该元素的文档中是唯一的，它是可选项。
- 包含内容列表：可以在简单类型中定义的元素类型，如表 4-2 所示。

表 4-2　简单类型元素

元　素	说　明
annotation	定义批注，用来对 XSD 文档中某一部分进行说明
restriction	为简单类型定义一个约束属性，即限制基类型
list	定义简单类型为列表类型
union	定义简单类型为联合类型

1. 限制类型

通过限制类型，可以从 XSD 的内置数据类型中继承，并添加一些约束条件形成一个新的类型，并且可以重复使用。

例如，在实际应用的数据库中，对用户名称的长度通常是有限制的，例如最长 20 个字符。而 XSD 的内置类型 string 则没有这个限制，所以这种类型用在这种场合就不大合适。但是，可以从 string 类型继承一个新的类型，并添加最大长度为 20 的限制，代码如下所示：

```
<xs:element name="用户名称" >
   <xs:simpleType >
      <xs:restriction base="xs:string">
         <xs:maxLength value="20"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

在这个示例中，我们从 string 类型通过受限继承的方式将定义的新的简单类型应用到“用户名称”元素上。restriction 元素可以包含一个或者多个子元素来限制基类型。这里的 maxLength 元素表示限制的条件为最大长度。

例如，下面的示例代码限制“用户名称”的值应该是长度在 6～20 之间的字符串。

```
<xs:element name="用户名称" >
   <xs:simpleType >
      <xs:restriction base="xs:string">
```

```
            <xs:maxLength value="20"/>
            <xs:minLength value="6"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

restriction 元素还包含一个很常用的 enumeration 元素，该元素可以将内容限制为一组可选择的值，也就是枚举限制。

下面的例子定义了带有一个限制的名为“usersType”的元素，该元素的取值可以是“新手上路”、“注册会员”、“金牌会员”或者“论坛元老”之一。

```
<xs:simpleType name="usersType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="新手上路"/>
        <xs:enumeration value="注册会员"/>
        <xs:enumeration value="金牌会员"/>
        <xs:enumeration value="论坛元老"/>
    </xs:restriction>
</xs:simpleType>
```

接下来，便可以将 usersType 类型应用到元素上，下面代码应用到了“会员级别”元素上。

```
<xs:element name="会员级别" type="usersType"/>
```

这里对限制长度的 maxLength 元素、minLength 元素以及枚举 enumeration 元素进行了简单介绍。其实，restriction 中还可以包含更多的限制条件，例如限制数值位数、限制字符串范围等，如表 4-3 所示给出了这些元素及其描述。

表 4-3　restriction 常用限制元素

限定元素	描　述
enumeration	定义可接受值的一个列表
fractionDigits	定义所允许的最大的小数位数，应用于继承自 decimal 的数据类型
length	定义类型值的固定长度。该值的长度单位依赖于基类型，对于 string 类型来说，单位是字符，对于 hexBinary 和 Base64Binary 来说，单位是 8 个字节。对于通过列表继承的类型来说，长度是列表中单元的个数
maxExclusive	定义排除时数值的上限，即所定义类型的值必须小于此值
maxInclusive	定义允许时数值的上限，即所定义类型的值必须小于或等于此值
maxLength	定义所允许的字符或者列表项目的最大长度
minExclusive	定义排除时数值的下限，即所定义类型的值必须大于此值
minInclusive	定义允许时数值的下限，即所允许类型的值必须大于或等于此值
minLength	定义所允许的字符或者列表项目的最小长度
pattern	定义所允许值应该遵循的表达式模式
totalDigits	定义所允许数字的最大个数
whiteSpace	定义空白字符(换行、回车、空格以及制表符)的处理方式，可选值有：preserve、replace 和 collapse。preserve 表示保留空白符，replace 表示所有空白都使用空格代替，collapse 表示首先进行 replace 操作，再把连续空格用一个空格代替，并删掉开头和结尾的空格

2. 列表类型

列表类型是指用空格隔开的内置数据类型列表。列表类型使用 xs:list 声明，itemType 属性用于指定元素的数据类型及列表内容。然后，在元素声明中引用该列表类型。下面给出的 XSD 架构示例就显示了内置数据类型和列表类型的用法。

例如，给出世界上的部分旅游景点名称，形成一个列表，然后在 XML 文档显示列表中属于中国景点的景点名称，其 XSD 架构文件如下所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="景点">
      <xs:restriction base="xs:string">
         <xs:enumeration value="泰山日出"/>
         <xs:enumeration value="桂林山水"/>
         <xs:enumeration value="雅典卫城"/>
         <xs:enumeration value="爱琴海"/>
         <xs:enumeration value="长江三峡"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="世界景点">
      <xs:list itemType="景点" />
   </xs:simpleType>
   <xs:simpleType name="中国景点">
      <xs:restriction base="世界景点">
         <xs:length value="3" />
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="景点名称" type="中国景点" />
</xs:schema>
```

在该 XSD 文档中，第一行是 XSD 标准声明，其中 xmlns 指定了该 XSD 包含的命名空间，文档中还定义了限制类型景点和列表类型景点。在内置数据类型定义中，xs:simpleType 声明这是一个包含简单类型元素的文档。xs:restriction 是一个限制基类型元素，它可以限制现有的简单类型的值以满足编写文档的需要，其属性 base 指定了数据类型为 string，说明元素内容只能包含字符串。在元素景点内部通过枚举给出了 5 个景点名称，在接下来的列表类型定义中，列表内容将取自前面给出的景点。

要列出“中国景点”的景点名称，在定义的简单类型元素景点名称中，首先指定其基类来自列表类型“世界景点”，然后指定其可以包含三个值；最后，声明 XML 文档元素景点名称，其值类型为“中国”。引用该 XSD 架构的 XML 文档如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<中国景点 xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
      xs:noNamespaceSchemaLocation="jingdian.xsd">
 泰山日出 桂林山水 长江三峡
</中国景点>
```

运行该 XML 文档，并对文档进行有效性检查，如图 4-3 所示。

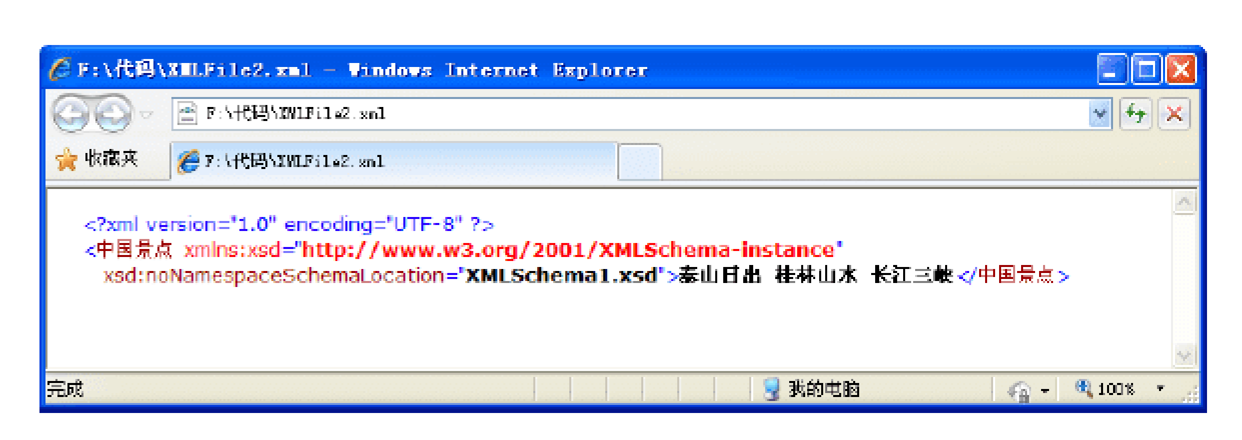


图 4-3　XML Schema 简单类型

3．联合类型

联合类型的值可以是内置类型，也可以属于列表类型。而且该类型可以包含多个内置数据类型或多个列表类型。联合类型由 xs:union 声明元素，使用 memberTypes 定义要包含的类型值。下面给出的 XSD 架构示例就显示了联合类型的用法：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="朝代称号">
      <xs:restriction base="xs:string">
         <xs:enumeration value="汉"/>
         <xs:enumeration value="宋"/>
         <xs:enumeration value="清"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="建立时间">
      <xs:restriction base="xs:integer">
         <xs:minInclusive value="960" />
         <xs:maxInclusive value="1644" />
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="朝代">
      <xs:list itemType="朝代称号" />
   </xs:simpleType>
   <xs:simpleType name="时间">
      <xs:union memberTypes="朝代 建立时间"/>
   </xs:simpleType>
   <xs:element name="朝代" type="时间" />
</xs:schema>
```

本例中，联合类型元素“时间”包含了一个 integer 类型的“建立时间”和一个列表类型的“朝代”。如下所示为符合该架构的 XML 文档内容：

```
<朝代>宋</朝代>
<朝代>960</朝代>
<朝代>清</朝代>
<朝代>1644</朝代>
```

4.3.2　实例描述

通过本节的学习，我们了解了简单类型又可以定义为三种：限制类型、列表类型和联合类

型，以及这些类型要怎样声明，这些知识只有通过不断的实践才能熟练掌握和应用。

图书馆里的书那么多，要想找到某一本书是很难的，于是图书管理员根据图书的用途将图书分门别类。这样，我们要找书是不是就容易多了。按照这种思路，我们运用今天学习的三种类型，也将我们的图书分分类吧。

4.3.3 实例应用

【例 4-2】 实现图书分类的 XSD

(1) 新建一个 XSD 文件，命名为“4-例 2.xsd”。

(2) 利用所学的知识，在 XSD 文件中定义图书列表、价格列表和图书分类。最终代码如下所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="图书">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Web 开发与应用"/>
            <xs:enumeration value="教你如何学编程"/>
            <xs:enumeration value="C#精编"/>
            <xs:enumeration value="ASP.NET 实战"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="价格">
        <xs:restriction base="xs:double">
            <xs:enumeration value="39.5"></xs:enumeration>
            <xs:enumeration value="45.0"></xs:enumeration>
            <xs:enumeration value="23.2"></xs:enumeration>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="工具书">
        <xs:list itemType="图书" />
    </xs:simpleType>
    <xs:simpleType name="报价">
        <xs:union memberTypes="工具书 价格">
        </xs:union>
    </xs:simpleType>
    <xs:element name="图书名称" type="报价" />
</xs:schema>
```

(3) 创建一个 XML 文件与 XSD 在同一目录下，命名为“4-例 2.xml”。

(4) 在 XML 中声明文档根元素为“图书名称”，它采用的值类型是“报价”，并引用 XSD。最终的 XML 文档代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<图书名称 xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="4-例 2.xsd">
教你如何学编程 ASP.NET 实战 C#精编
```

```
</图书名称>
```

(5) 保存对 XML 文档的修改，实例就完成了。

4.3.4　运行结果

在 Web 中运行该 XML 文档，并对 XML 文档进行有效性检查，如图 4-4 所示。

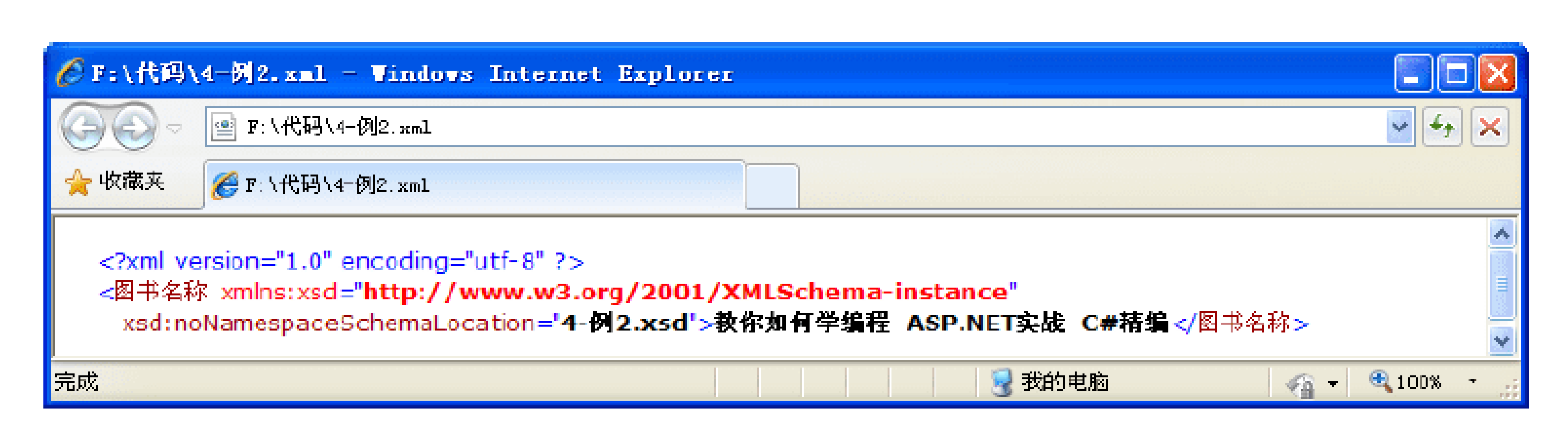


图 4-4　XSD 简单类型

4.3.5　实例分析



源码解析：

在本实例中，XSD 文件的第一行指定了该 XSD 引用的命名空间。我们定义了限制类型的“图书”和“价格”，在元素图书内通过枚举给出了 4 本图书的名称。列表类型“工具书”的内容将取自前面给出的图书，“报价”则指定为联合类型。

要注意：XML 和 XSD 引用的命名空间不同，其中 XSD 引用的命名空间为“http://www.w3.org/2001/XMLSchema”；XML 引用的命名空间为“http://www.w3.org/2001/XMLSchema-instance”。

4.4　定义匿名类型

严格来说，匿名类型不属于一种特定的数据类型。它只是在定义元素类型时的一种编写方式，下面我们就来了解一下匿名类型的定义方法。



视频教学：光盘/videos/04/匿名类型.avi　　长度：3 分钟

在前面的例子中，每个定义的类型都有唯一的名称。例如，如下代码定义了一个名为“password”的简单类型。

```
<xs:simpleType name="password">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
</xs:simpleType>
```

password 类型可接受的值是由 8 个字符组成的一行字符，而且这些字符必须是大写或小写字母 a～z 或者数字 0～9。

这样一来，password 类型就可以在 XSD 中多次使用。而且应用该类型的元素可以出现在类型定义之前或者之后，相对位置并不重要。例如，如下的代码定义“密码”元素的值为 password 类型：

```
<xs:element name="密码" type="password" />
```

但是，为类型定义一个名称并不是必需的。我们可以将类型的定义作为元素声明的一部分，而无须再为类型定义名称。我们将这种情况下声明的类型称为匿名类型。

例如，下面是采用匿名类型声明密码元素的代码：

```
<xs:element name="密码">
   <xs:simpleType >
      <xs:restriction base="xs:string">
         <xs:pattern value="[a-zA-Z0-9]{8}"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

在这个例子中，element 元素的声明没有 type 属性，而且 simpleType 类型中也没有 name 属性。如果文档中只有一处会出现某个元素的声明，则可以使用这样的匿名类型定义。

当然，复杂类型也可以用相同的方法来定义匿名类型，例如下面给出的示例代码。

```
<xs:element name="主要参数">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="体积" type="xs:string" />
         <xs:element name="通话时间" type="xs:string" />
         <xs:element name="屏幕参数" type="xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

4.5 专 辑 信 息

参照 4.3.1 节讲解简单类型时描述手机信息的 XML 文档。我们可以了解到，复杂类型不仅可以包含字符内容，还可以包含子元素和元素属性。现在，就让我们来好好学习一下吧。

复杂类型可以包含子元素和属性，而简单类型则不能。简单类型通常用来表示大多数平台所支持的基础数据类型，像整型、字符串、日期和时间以及布尔，而复杂类型则用来表示数组、集合、结构和自定义的类型。


视频教学：光盘/videos/04/专辑信息.avi　　长度：15 分钟

4.5.1　基础知识——复杂数据类型

在 XSD 中将包含子元素和元素属性的元素称为复杂类型。复杂类型根据包含的内容，可以分为 4 种：简单类型、纯元素类型、混合类型和空类型。

复杂类型由 complexType 定义，语法如下：

```
<complexType name = 名称 id = ID mixed = Boolean : false >
   <annotation | simpleContent | complexContent | group | all | choice | sequence
| attribute | attributeGroup | anyAttribute >…
</complexType>
```

语法中各个参数的含义如下。

- name：复杂类型的名称，该名称必须是在 XML 命名空间规范中定义的无冒号名称。
- id：该元素的 ID。id 值必须属于类型 ID 并且在包含该元素的文档中是唯一的。它是可选项。
- mixed：一个指示符，指示是否允许字符数据出现在该复杂类型的子元素之间。默认值为 false。如果 simpleContent 元素是子元素，则不允许 mixed 属性。如果 complexContent 元素是子元素，则该 mixed 属性可被 complexContent 元素的 mixed 属性重写。它是可选项。
- 包含内容：元素内容均可以定义在复杂类型中，如表 4-4 所示。

表 4-4　复杂类型元素

元　素	描　述
annotation	批注，只在 XSD 中起注释的作用，不影响整体内容
simpleContent	从简单类型派生复杂类型。用于声明包含字符内容和属性的元素，但不包含子元素的声明
complexContent	从复杂类型派生出新的复杂类型。用于声明包含属性和子元素的元素声明，但不包含字符数据的元素声明
group	将若干元素声明归为一组，将它们作为一个组并入复杂类型定义
all	表示符合元素声明的所有元素都应该出现且只能出现一次
choice	用来声明只有一个相容元素必须出现，用于选择情况
sequence	定义了一系列元素并且必须按照模式中定义的顺序显示，所有子元素如果在实例文档中出现，则必须按照该顺序显示
attribute	为出现的元素定义属性
attributeGroup	定义属性列表
anyAttribute	在 XML 文档中添加未被 Schema 指定过的属性

下面我们给出一个 XML 文档，要求根据该文档给出文档的 XSD 架构。XML 文档代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<电影 xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="4-6-1.xsd"
     语言="英语">
    <电影名 imdb 编码="tt0209163">木乃伊归来</电影名>
    <主演>布兰登·弗雷泽</主演>
    <导演>斯蒂芬·索莫斯</导演>
    <简介>故事发生在 1933 年，也就是蝎子王的年代。距上次瑞克-奥克康纳(布兰登-弗雷泽饰)和无惧的埃及考古学家艾弗琳(瑞切尔-薇茨饰)奋力与 3000 岁的“老怪”伊默特普(阿诺-沃斯洛饰)作战得以逃生，已有 8 年了……</简介>
    <出品信息>
        <出品公司>环球公司</出品公司>
        <上映时间>2001-05-04</上映时间>
  </出品信息>
    <类型 影片类型="动作片|冒险片|喜剧片|爱情片|科幻片|恐怖片">
        <冒险片 />
    </类型>
</电影>
```

为了详细说明复杂类型各个元素的含义及用法，以及复杂类型 4 种内容类型的区别，下面我们将以“电影”信息为例详细说明。

1. 简单类型

在上节我们学到，简单类型只包含文本的元素，如果要在复杂类型里定义简单类型元素必须添加 simpleContent 元素。在 simpleContent 元素内定义扩展或限定简单类型元素，语法如下：

```
<xs:element name="某个名称">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension|restriction base="basetype">
        ....
        ....
      </xs:extension|restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```



在定义扩展或限定简单类型元素时，必须使用 extension 或 restriction。extension 元素表示对 simpleContent 的扩展；restriction 元素表示对 simpleContent 的约束。

根据该定义，我们可以给出“电影”的部分 XSD 文档，如下所示：

```
<xs:element name="电影名">
  <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="imdb 编码" type="xs:string" />
         </xs:extension>
      </xs:simpleContent>
```

```
    </xs:complexType>
</xs:element>
```

在此实例中，attribute 元素表示为出现的元素定义属性，“电影名”包含属性“imdb 编码”，所以为复杂类型。但是元素中不再包含子元素，而只包含字符数据，所以可以用 simpleContent 元素声明简单类型内容。我们使用 extension 元素为“电影名”定义属性，表示对“电影名”进行扩展。

2．纯元素类型

纯元素类型，顾名思义就是仅包含元素的元素类型。如下所示 XML 文档中，“电脑”仅包含了 4 个纯元素类型的子元素。

```
<电脑>
    <显示器>三星</显示器>
    <主机>盘古系列</主机>
    <鼠标>双飞燕</鼠标>
    <键盘>双飞燕</键盘>
</电脑>
```

相对应的，在 XSD 中的定义就可以，如下所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="电脑">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="显示器" type="xs:string" />
                <xs:element name="主机" type="xs:string" />
                <xs:element name="鼠标" type="xs:string" />
                <xs:element name="键盘" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

在该 XSD 文档中，sequence 表示定义了一系列元素并且必须按照模式中指定的顺序显示。我们已经定义了“电脑”中子元素的顺序为“显示器”→“主机”→“鼠标”→“键盘”。在 XML 文档中，这些子元素如果出现，则必须按照此顺序出现，否则，将会出错。

```
<xs:element name="电影">
  <xs:complexType>
      <xs:sequence>
    </xs:complexType>
</xs:element>
```

于是，根据纯元素类型的特点，我们给出了“电影”的部分 XSD 文档，如下所示：

```
<xs:element name="电影">
     <xs:complexType>
          <xs:sequence>
              …
```

```
                <xs:element name="主演" type="xs:string"/>
                <xs:element name="导演" type="xs:string"/>
                <xs:element name="简介" type="xs:string"/>
                </xs:element>
                …
            </xs:sequence>
        </xs:complexType>
</xs:element>
```

这里的“主演”、“导演”和“简介”都是作为混合类型中的纯元素类型出现的。

3. 混合类型

混合类型包含了属性、元素及文本。这里，我们通过对 group、all 和 choice 的讲解来学习混合类型的使用方法。

1) group

group 元素是将若干元素声明归为一组，以便将它们当作一个组并入复杂类型定义，如下段代码所示：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="显示器" type="xs:string" />
   <xs:element name="主机" type="xs:string" />
   <xs:element name="鼠标" type="xs:string" />
   <xs:element name="键盘" type="xs:string" />
   <xs:attribute name="类型" type="xs:string" />
   <xs:group name="配件">
      <xs:sequence>
         <xs:element ref="显示器" />
         <xs:element ref="主机" />
         <xs:element ref="键盘" />
      </xs:sequence>
   </xs:group>
   <xs:element name="电脑">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="配件">
               <xs:complexType>
                  <xs:group ref="配件" />
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute ref="类型"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

在该例中，使用 group 将“显示器”、“主机”和“键盘”归为一个组，在“配件”中使用 ref 指向该组。

根据该例，可以知道，group 的语法如下：

```
<group name=名称 id = ID maxOccurs=数值 minOccurs=数值 ref=组名>
  <annotation|all | choice | sequence> …
</group>
```

- maxOccurs 和 minOccurs 分别是允许元素出现的最多和最少次数。
- ref 是在该架构(或由指定的命名空间指示的其他架构)中声明的组的名称。ref 值必须是组名称。ref 可以包含命名空间前缀。如果 ref 属性出现，则 id、minOccurs 和 maxOccurs 可以出现。Ref 和 name 是互相排斥的。

根据 XSD 文档，我们可以得到 XML 文档，代码如下：

```
<电脑 类型="台式机">
    <配件>
      <显示器/ >
      <主机/ >
      <键盘</ >
    </配件>
</电脑>
```

2)　all

all 元素表示符合元素声明的所有元素都应该出现(以任何顺序)且只能出现一次。all 的语法结构如下：

```
<all id = ID maxOccurs=数值 minOccurs= 数值>
  <annotation| element>…
</all>
```

如下列代码展示了 all 的用法：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="电脑">
        <xs:complexType>
            <xs:all>
                <xs:element name="显示器" type="xs:string" />
                <xs:element name="主机" type="xs:string" />
                <xs:element name="鼠标" type="xs:string" />
                <xs:element name="键盘" type="xs:string" />
            </xs:all>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

该例中，使用 all 定义了子元素都应该出现，并且只能出现一次，对应的 XML 文档如下：

```
<电脑>
    <显示器>三星</显示器>
    <主机>盘古系列</主机>
    <键盘>双飞燕</键盘>
    <鼠标>双飞燕</鼠标>
</电脑>
```

3) choice

choice 用来声明只有一个相容元素必须出现，用于选择情况。语法结构如下：

```
<choice id = ID maxOccurs=数值 minOccurs=数值>
   <annotation | element | group | choice | sequence | any> …
</choice>
```

还以“电脑”为例，使用 choice 元素为电脑进行分类选择，代码如下：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="电脑">
      <xs:complexType>
         <xs:all>
            <xs:element name="显示器" type="xs:string" />
            <xs:element name="主机" type="xs:string" />
            <xs:element name="鼠标" type="xs:string" />
            <xs:element name="键盘" type="xs:string" />
            <xs:element name="类别">
               <xs:complexType>
                  <xs:choice>
                     <xs:element name="笔记本" type="xs:string"/>
                     <xs:element name="台式机" type="xs:string"/>
                     <xs:element name="组装机" type="xs:string"/>
                  </xs:choice>
               </xs:complexType>
            </xs:element>
         </xs:all>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

我们使用 choice 元素为“电脑”的子元素“种类”定义了三个选项，在 XML 文档中必须选择一项，XML 文档如下：

```
<?xml version="1.0" encoding="utf-8"?>
<电脑>
   <显示器>三星</显示器>
   <主机>盘古系列</主机>
   <键盘>双飞燕</键盘>
   <鼠标>双飞燕</鼠标>
   <类别>
      <组装机 />
   </类别>
</电脑>
```

4) 空类型

空类型不能包含内容，只能包含属性。如下列 XML 表示一个空的 XML 元素：

```
<产品型号 编号="007" />
```

上面的“产品型号”除了属性“编号”外，根本没有内容。为了定义无内容的类型，我们就必须声明一个在其内容中只包含属性的类型，而事实上我们不需要声明任何包含其他内容的元素，XSD 文档如下所示：

```
<xs:element name="产品型号">
   <xs:complexType>
      <xs:attribute name="编号" type="xs:positiveInteger"/>
   </xs:complexType>
</xs:element>
```

该例中，我们定义了名为“产品型号”的复合类型，并且声明了“编号”属性，且没有任何元素内容。

到了这里，复杂类型的知识已经学习完毕了。那么，对于本章节开始的“电影”实例，根据提供的 XML 文档得到相应的 XSD 文档也不是难事了，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="电影">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="电影名">
               <xs:complexType>
                  <xs:simpleContent>
                     <xs:extension base="xs:string">
                        <xs:attribute name="imdb编码" type="xs:string" />
                     </xs:extension>
                  </xs:simpleContent>
               </xs:complexType>
            </xs:element>
            <xs:element name="主演" type="xs:string"/>
            <xs:element name="导演" type="xs:string"/>
            <xs:element name="简介" type="xs:string"/>
            <xs:element name="出品信息" minOccurs="0" maxOccurs="1">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="出品公司" type="xs:string"/>
                     <xs:element name="上映时间" type="xs:date"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="类型">
               <xs:complexType>
                  <xs:choice>
                     <xs:element name="动作片" type="xs:string"/>
                     <xs:element name="冒险片" type="xs:string"/>
                     <xs:element name="喜剧片" type="xs:string"/>
                     <xs:element name="爱情片" type="xs:string"/>
                     <xs:element name="科幻片" type="xs:string"/>
                     <xs:element name="恐怖片" type="xs:string"/>
```

```
                    </xs:choice>
                    <xs:attribute name="影片类型" type="xs:string" />
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="语言" type="xs:string"/>
    </xs:complexType>
</xs:element>
</xs:schema>
```

4.5.2 实例描述

我们都喜欢听音乐，一首好听的专辑包含的信息一般都在专辑的封面上。今天，我们要使用 XML 和 XSD 将专辑信息表达出来，使之更加生动形象。

4.5.3 实例应用

【例 4-3】 专辑信息

(1) 新建一个名为“4-例 3.XSD”的 XSD 文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="专辑">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="歌曲名" type="xs:string"/>
                <xs:element name="演唱者" type="xs:string"/>
                <xs:element name="作词" type="xs:string"/>
                <xs:element name="作曲" type="xs:string"/>
                <xs:element name="MV" minOccurs="0" maxOccurs="1">
                    <xs:complexType>
                        <xs:all>
                            <xs:element name="导演" type="xs:string"/>
                            <xs:element name="主演" type="xs:string"/>
                        </xs:all>
                    </xs:complexType>
                </xs:element>
                <xs:element name="类型">
                    <xs:complexType>
                        <xs:choice>
                            <xs:element name="摇滚" type="xs:string"/>
                            <xs:element name="Hip-Hop" type="xs:string"/>
                            <xs:element name="电子乐" type="xs:string"/>
                            <xs:element name="古典" type="xs:string"/>
                        </xs:choice>
                        <xs:attribute name="歌曲类型" type="xs:string" />
```

```
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="语言" type="xs:string"/>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

(2)　新建一个 XML 文档，并引用 XSD 文档“4-例 3”，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<专辑 xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="4-例 3.xsd"
     语言="中文">
    <歌曲名>半城烟沙</歌曲名>
    <演唱者>许嵩</演唱者>
    <作词>许嵩</作词>
    <作曲>许嵩</作曲>
    <MV>
        <导演>未知</导演>
        <主演>未知</主演>
    </MV>
    <类型 歌曲类型="摇滚|Hip-Hop|电子乐|古典">
        <古典 />
    </类型>
</专辑>
```

(3)　保存，并将 XSD 文档和 XML 文档放在同一目录下。

4.5.4　运行结果

在浏览器中打开 XML 文档和 XSD 文档，效果如图 4-5 所示。

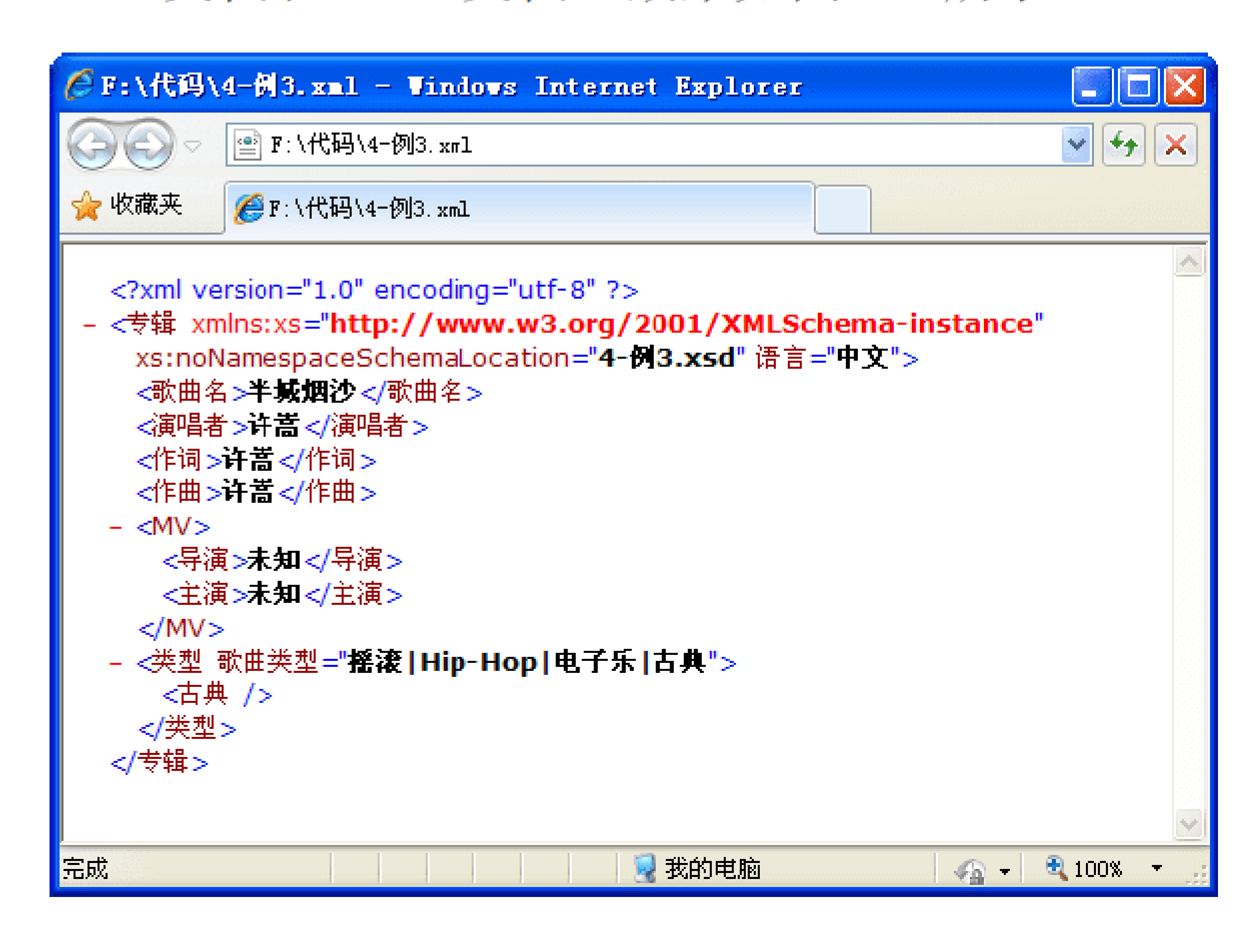


图 4-5　XML 文档效果

4.5.5　实例分析



源码解析：

在该实例中，使用 sequence 元素确保声明的子元素都会出现在 XML 文档中。使用 all 元素表示符合元素声明的元素只能出现一次。使用 choice 元素用来进行选择。使用 attribute 定义一个元素的属性。

4.6　构造 XSD 的元素和属性

前面的小节中对 XSD 的两种数据类型进行了详细的讲解。通过大量的实例，读者可能已经掌握了 XSD 文档的基本构造方法。本节将对 XSD 架构文档中的细节进行介绍，首先是如何声明一个元素。



视频教学：光盘/videos/04/XSD 的元素和属性.avi　　长度：12 分钟

4.6.1　基础知识——声明元素

在一个 XSD 文档中使用 schema 来声明根元素，根元素指定文档类型，包括模式的约束、XML 模式名称空间定义、版本信息和语言信息等。

根元素在每个文档中只能出现一次。要声明一个普通元素需要使用 element 来定义，它可以在文档中出现多次。

element 定义元素的详细语法格式如下：

```
<xsd:element abstract | block | default | final | fixed | form | id | maxOccurs
| minOccurs | name | nillable | ref | substitutionGroup | type >
       <xsd:annotation | xsd:simpleType | xsd:complexType | xsd:unique |
xsd:key | xsd:keyref >…
</xsd:element>
```

语法中，abstract 和 block 等是 element 元素的属性，在表 4-5 中给出了这些属性的说明。

表 4-5　element 元素的属性说明

属 性 名	说　明
abstract	一个 boolean 型的值，指定元素是否可以在实例文档中使用。默认值为 false
block	block 属性用于防止具有指定派生类型的元素替代该元素
default	为元素指定一个默认值
final	设置 element 元素上 final 属性的默认值
fixed	为元素指定一个固定值，并且是不可更改的值

续表

属 性 名	说　明
form	指定元素的形式。默认值是 schema 元素 elementFormDefault 属性的值，必须是 qualified 或者 unqualified。如果该值是非限定的，则无须通过命名空间前缀限定该元素。如果该值是限定的，则必须通过命名空间前缀限定该元素
id	指定元素的 ID，该值必须属于类型 ID，并且在包含该元素的文档中是唯一的
maxOccurs	指定元素可以在包含元素中出现的最大次数
minOccurs	指定元素可以在包含元素中出现的最小次数
name	指定元素的名称
nillable	一个 boolean 型的值，指定是否可以将显式的零值分配给该元素。默认值为 false
ref	在架构中声明元素的名称。该值必须是限定名称，可以包含命名空间前缀
substitutionGroup	指定可用来替代该元素的元素名称。该元素必须具有相同的类型或者是从指定元素类型派生的类型
type	指定元素数据类型，该数据类型可以是内置数据类型，或者是在此架构中定义的 simpleType 或者 complexType 元素

下面，我们来了解一下 element 元素的子元素，其中一些在前面已经用过，如表 4-6 所示。

表 4-6　element 元素的子元素说明

元 素 名	说　明
annotation	定义批注
simpleType	定义一个简单类型
complexType	定义一个复杂类型
unique	指定属性或者元素值在指定范围内必须是唯一的
key	指定属性或者元素值必须是指定范围内的键。键的范围为实例文档中包含的 element。键意味着数据在指定范围内应是唯一的、不为零的并且始终存在的
keyref	指定属性或元素值(或一组值)与指定的 key 或 unique 元素的值相对应



对于表 4-5 和表 4-6 中的数据，读者可能会觉得一时难以理解，其实列出 element 元素全部的属性及子元素只是为了让读者有一个初步的了解，以方便后面阅读一些复杂的 Schema 文档。在本书中，也只用到了一些比较常用的属性和子元素。

例如，我们有一个保存游戏信息的 XML 文档，包括游戏类型、名称、开发者和版本，它的内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<游戏列表>
    <游戏 类型="益智类">
        <名称>推箱子</名称>
        <开发者>汇智科技</开发者>
        <版本>1.1.5.3RC</版本>
    </游戏>
```

```
    <游戏 类型="射击类">
        <名称>穿越火线</名称>
        <开发者>昌利数码</开发者>
        <版本>2.1.2RC</版本>
    </游戏>
    <游戏 类型="休闲类">
        <名称>超级大富翁</名称>
        <开发者>豆包娱乐</开发者>
        <版本>1.0.6</版本>
    </游戏>
</游戏列表>
```

在这个 XML 文档中，根元素游戏列表只有一个游戏子元素，但是每个游戏元素中包含的子元素和属性都不同。游戏元素包含的子元素有名称、开发者和版本，以及一个类型属性。

下面，创建相应的 XSD 架构文件。首先使用 schema 定义根元素及命名空间，然后针对游戏列表和游戏添加相应的元素定义语句。最终形成的完整架构如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="游戏列表">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" name="游戏">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="名称" type="xs:string" />
                            <xs:element name="开发者" type="xs:string" />
                            <xs:element name="版本" type="xs:string" />
                        </xs:sequence>
                        <xs:attribute name="类型" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

如上述代码所示，使用 element 元素定义根元素为游戏列表。另外，由于它包含有子元素及属性，所以使用 complexType 声明为复杂类型，然后按照元素在文档中出现的先后顺序进行定义。

再往下定义游戏元素及其子元素，还有类型属性。其中“maxOccurs="unbounded"”定义游戏元素至少出现一次，无次数限制。类型属性的“use="required"”表示在 XML 文档中必须出现，且它的值是一个枚举值。如下定义了枚举数据类型的“游戏类型”，它的值是一些字符串组成的列表：

```
 <xs:simpleType name="游戏类型">
     <xs:restriction base="xs:string">
```

```
        <xs:enumeration value="益智类"/>
        <xs:enumeration value="射击类"/>
        <xs:enumeration value="休闲类"/>
        <xs:enumeration value="策略类"/>
        <xs:enumeration value="冒险类"/>
    </xs:restriction>
</xs:simpleType>
```

元素 simpleType 定义了一个简单类型的“游戏类型”，restriction 元素限制其内容为 string 类型。要列举所有游戏类型，这里用到了 enumeration 元素，其 value 属性指定了枚举值。当使用枚举类型时，元素的值必须从组值中选择，且只能选择一个。

对 XSD 架构内“类型”属性的类型进行修改，应用上面创建的枚举类型“游戏类型”，代码如下所示：

```
<xs:attribute name="类型" type="游戏类型" use="required" />
```

当架构生成后，在 XML 文档中引用该架构，并运行文档进行有效性检验，如图 4-6 所示。

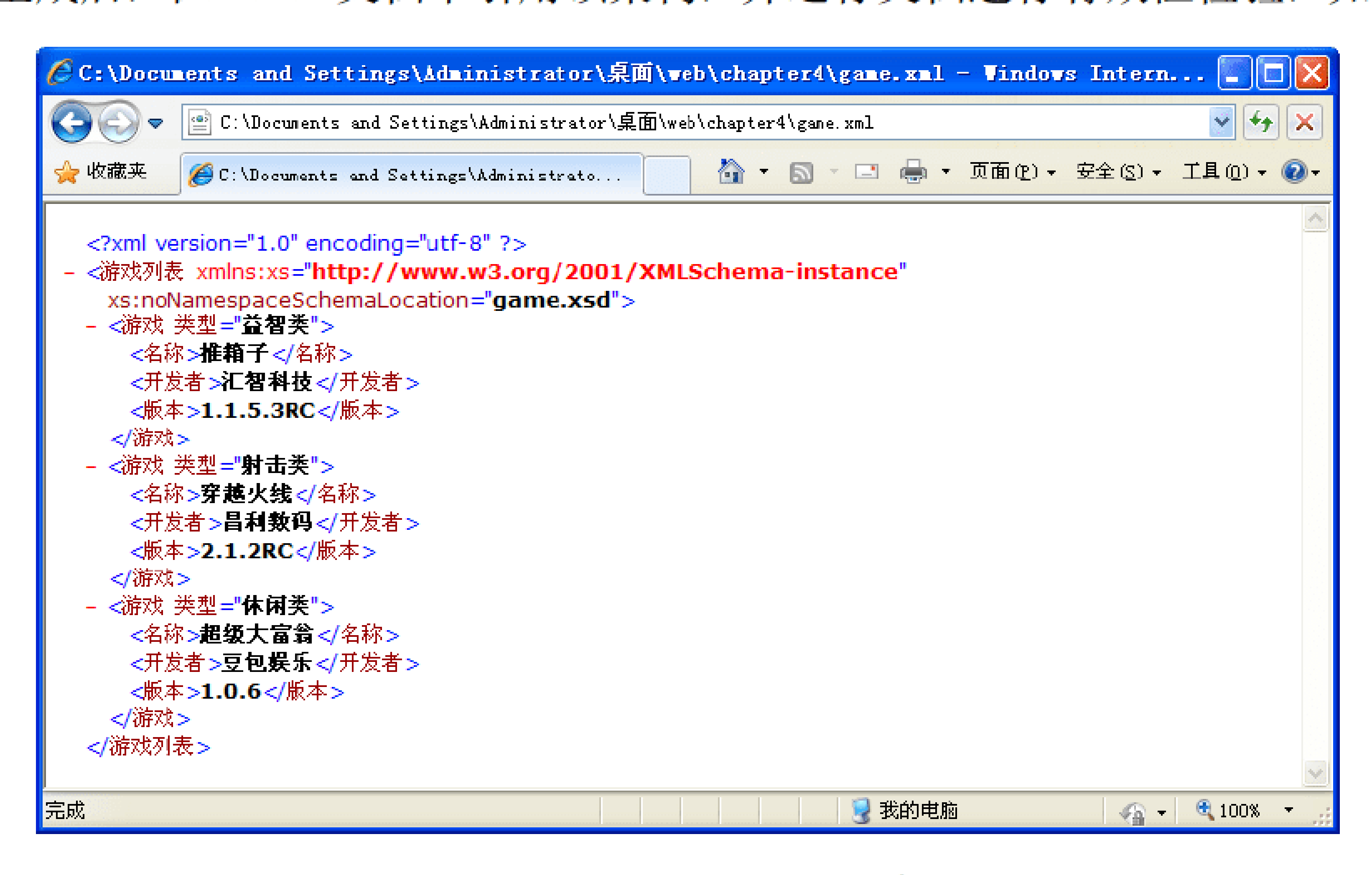


图 4-6　XML Schema 元素声明

4.6.2　基础知识——声明属性

关于如何在 XSD 中声明属性，其实在本节之前就已经使用过，这里我们来详细了解一下。XML 文件中的属性可以使用 attribute 元素来声明，attribute 元素的详细语法格式如下：

```
<xsd:attribute default | fixed | form | id | name | ref | type | use >
     <xsd:annotation | xsd:simpleType >…
</xsd:attribute>
```

<xsd:attribute>元素中的各属性及子元素如表 4-7 和表 4-8 所示。

表 4-7　attribute 元素的属性说明

属　性	说　明
default	指定属性的默认值
fixed	指定属性的固定值
form	指定属性的格式。默认值是包含该属性的 schema 元素的 attributeFormDefault 属性的值
id	指定元素的 ID。ID 值必须属于类型 ID，并且在包含该元素的文档中是唯一的
name	指定属性的名称
ref	在该架构中声明属性的名称。ref 值必须是限定名，类型可以包括命名空间前缀。但是要注意 name 和 ref 属性不能同时出现
type	指定属性的数据类型
use	指示如何使用属性。有效值是 optional(属性是可选的并且可以具有任何值)、prohibited(该属性用于其他复杂类型的限制中以禁止使用现有属性)和 required(该属性是必需的，可以是 type 属性允许的任何值)

表 4-8　attribute 元素的子元素说明

子 元 素	说　明
annotation	定义批注
simpleType	定义一个简单类型

表 4-7 和表 4-8 详细介绍了如何声明属性，以及各个选项的作用。下面通过一个 XSD 架构文档来看看属性的具体应用，代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="订餐清单">
      <xs:complexType>
         <xs:sequence>
            <xs:element maxOccurs="unbounded" name="餐桌">
               <xs:complexType mixed="true">
                  <xs:sequence minOccurs="0">
                     <xs:element maxOccurs="unbounded" name="菜单"
                         type="xs:string" />
                     <xs:element name="可乐" type="xs:string" />
                  </xs:sequence>
                  <xs:attribute name="编号" type="xs:unsignedByte"
                      use="required" />
                  <xs:attribute name="状态" type="xs:string" use="required"/>
                  <xs:attribute name="价格" type="xs:unsignedByte"
                    use="optional" default="0" />
               </xs:complexType>
            </xs:element>
         </xs:sequence>
```

```
        </xs:complexType>
    </xs:element>
</xs:schema>
```

可以看到，在这里使用 attribute 元素定义了 3 个属性：编号、状态和价格。其中，编号和状态属性都使用“use="required"”语句定义为必须具有的；价格属性的“use="optional"”定义为可选的，并且“default="0"”语句指定该属性的默认值为 0。

现在，针对上面的 XSD 架构来创建一个符合规范的 XML 文件，它的内容应该类似如下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<订餐清单 >
    <餐桌 编号="1" 状态="空">
    </餐桌>
    <餐桌 编号="2" 状态="正在用餐" 价格="98">
        <菜单>鱼香肉丝</菜单>
        <菜单>招牌水煮鱼</菜单>
        <菜单>地三鲜(大份)</菜单>
        <可乐>4 瓶</可乐>
    </餐桌>
</订餐清单>
```

为了进行验证，在 XML 中加入对 XSD 架构的引用。再运行 XML 文档，效果如图 4-7 所示。

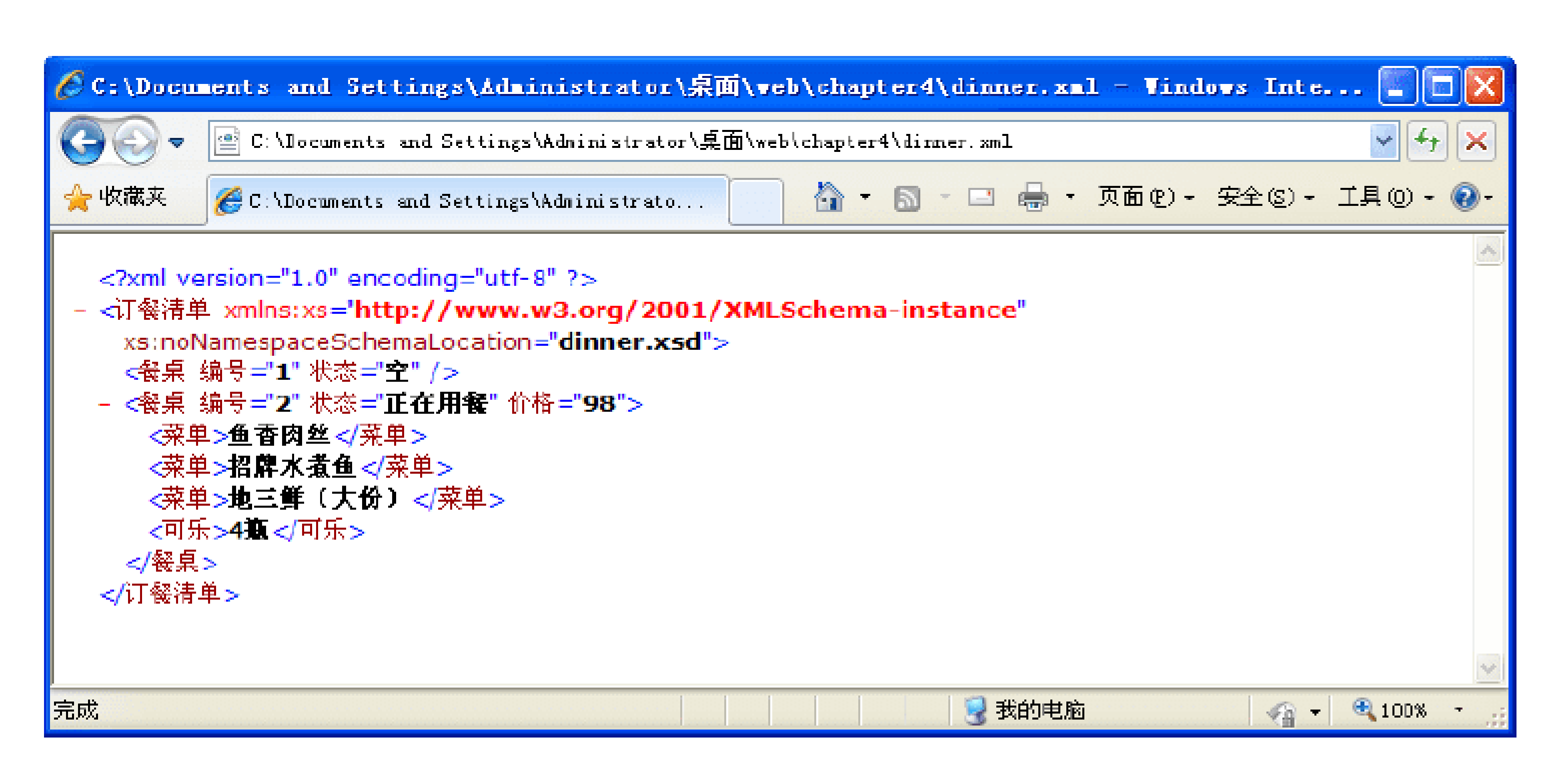


图 4-7　XML Schema 属性声明

对于一个元素的多个属性声明，可以使用 attributeGroup 元素将它们定义在一个属性组中，然后再通过 ref 元素将属性组链接到所属元素的属性声明中，这样做可以简化对架构文档的维护及更新操作。

例如，下面是给出的示例代码：

```
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="公交车">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" name="车">
                    <xs:complexType>
```

```
                    <xs:attributeGroup ref="规格"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:attributeGroup name="规格">
    <xs:attribute name="长" type="xs:string" use="required"/>
    <xs:attribute name="宽" type="xs:string" use="required"/>
    <xs:attribute name="高" type="xs:string" use="required"/>
  </xs:attributeGroup>
</xs:schema>
```

引用该 XSD 架构的 XML 文档如下所示：

```
<公交车>
  <车 长="4m" 宽="3m" 高="2m" >6 路</车>
  <车 长="4m" 宽="3m" 高="2.6m" > 9 路</车>
</公交车>
```

4.7 编写程序验证 XSD 文档

XSD 架构不仅可以用作 XML 文档的规范，还可以用来检验 XML 文档的合法性。有些检验程序能够根据 XSD 架构检验 XML 文档，并报告错误。这样的程序分为两种，一种是解析器，另一种是编辑器。在创建 XML 和 XSD 时通常会使用编辑器，如 VS2010，这时候它可以用来检验 XML 文档。而在运行程序时，则可以使用解析器对 XML 文档进行合法性检查。

**视频教学**：光盘/videos/04/验证 XSD 文档.avi　　长度：5 分钟

4.7.1 基础知识——指定 XSD 位置

如图 4-8 所示为使用校验程序根据 XSD 架构对 XML 文档进行校验的过程。

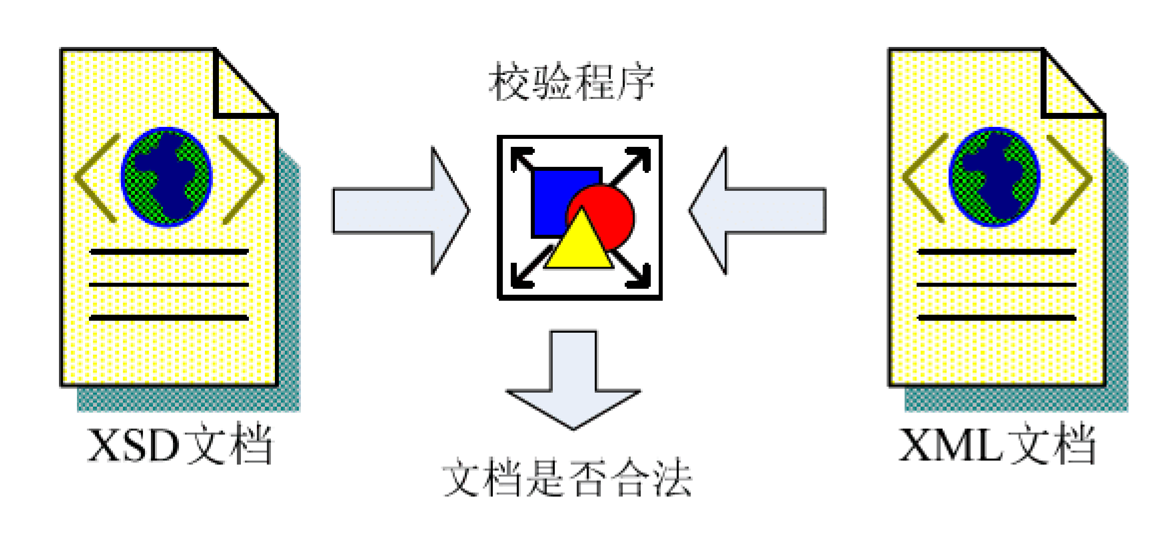


图 4-8 校验过程

检验程序首先需要知道使用哪个大纲来验证文档，XML 文档可以使用标准的方法来指定它所遵循的规范。根据实例文档中有没有使用命名空间，可以选择使用 schemaLocation 或者

noNamespaceSchemaLocation 来指定架构。这两个属性都属于 XML 规范命名空间。

如果验证的元素或者属性不属于任何命名空间，则使用 noNamespaceSchemaLocation 属性来指定 XSD 文档的物理位置，否则使用 schemaLocation 来指定。

例如，下面的 XML 文档使用的 XSD 架构位于“D:\XML\schemas\banks.xsd”。

```
<!-- 使用 schemaLocation 指定 XSD 架构的位置  -->
<附近银行 xmlns="http://www.abclife.com/banks"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.abclife.com/banks
D:\XML\schemas\banks.xsd">
    <银行>
        <名称>中国工商银行</名称>
        <地址>人民路与文化路交叉口</地址>
        <类型>分行</类型>
    </银行>
    <银行>
        <名称>交通银行</名称>
        <地址>北京华联南门西侧</地址>
        <类型>ATM</类型>
    </银行>
</附近银行>
```

需要注意的是，这个文档的默认命名空间和 XSD 的目标命名空间相同。还需要注意的是，属性 schemaLocation 属于 XML 实例命名空间。schemaLocation 的值是一串用空格分开的命名空间和 XSD 架构序列。例如，在本实例中可以理解为“使用位于 D:\XML\schemas\banks.xsd 的架构来验证属于 http://www.abclife.com/banks 命名空间中的元素”。

下面的示例演示了当一个文档中的元素来自两个命名空间时如何指定各自的架构。

```
<!-- 存在多个命名空间时使用 schemaLocation -->
<开始菜单 xmlns="http://www.mypc.com.cn/menus"
     xmlns:set="http://www.mypc.com.cn/settings"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.mypc.com.cn/menus
D:\XML\schemas\menus.xsd http://www.mypc.com.cn/settings
D:\XML\schemas\settings.xsd">
    <程序>Microsoft Office Word</程序>
    <程序>Microsoft Visual Studio</程序>
    <实用软件>PDF 阅读器</实用软件>
    <实用软件>万能视频播放器</实用软件>
    <set:设置 显示大图标="false" 显示数量="5" 所属用户="somboy">
    </set:设置>
</开始菜单>
```

这个例子中的设置元素属于命名空间 http://www.mypc.com.cn/settings，schemaLocation 属性把它和“D:\XML\schemas\menus.xsd”架构关联起来。

现在看来，架构的位置实际上就是一个 URL。所以，我们也可以使用网络上的架构，只需通过 URL 来引用它即可。例如，如下的代码演示了这种方法：

```
<!-- 使用位于网络位置的 XSD 架构 -->
<收藏夹 xmlns="http://www.ayhncn.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ayhncn.com
http://www.itzcn.com/favorite.xsd">
    <收藏>
        <名称>窗内网</名称>
        <地址>www.itzcn.com</地址>
    </收藏>
    <收藏>
        <名称>汇智科技</名称>
        <地址>www.hztec.net</地址>
    </收藏>
</收藏夹>
```

这个例子中把命名空间 http://www.ayhncn.com 映射到架构 http://www.itzcn.com/favorite.xsd。只是需要注意的是，虽然它们都是 URL，但是 http://www.ayhncn.com 只是一个简单的命名空间，而 http://www.itzcn.com/favorite.xsd 才是一个指向实际内容(XSD 架构)的 URL。

4.7.2　实例描述

虽然在编写 XML 文档时通过指定 XSD 架构的位置可以进行校验，但并不是所有情况下都需要 XML 编辑器。作为一名优秀的开发人员，习惯用程序的方式来解决这个问题。下面，我们就通过 ASP.NET 编写一个 Web 网页来实现 XSD 架构的校验。

4.7.3　实例应用

【例 4-4】 编写程序验证 XSD 文档

(1) 首先在 Visual Studio 2010 中新建一个 Web 应用程序。

(2) 在默认页面中创建一个表单允许用户选择 XML 文件和 XSD 文件，输入命名空间。在表单中还有一个提交按钮以及一个显示结果的标签。最终代码如下所示：

```
    <h2>欢迎使用文档校验程序</h2>
    <p>
    选择 XML 文件位置:    <asp:fileupload runat="server"
id="xmlFile"></asp:fileupload><br />
    选择 XSD 文件位置:    <asp:fileupload runat="server"
id="xsdFile"></asp:fileupload><br />
    命名空间: <asp:TextBox ID="txtNameSpace" runat="server"></asp:TextBox>如果
没有，请保留为空。<br />
    <asp:Button ID="Button1" runat="server" Text="开始校验"
     onclick="Button1_Click" />
    <hr/>
        <span style="color:#FF0000;">
        <asp:Literal ID="ltResult" runat="server"/>
```

```
        </span>
    </p>
```

(3) 这样前台页面就制作好了。开始编写实现代码，先转到后台文件，添加对 System.Xml 和 System.Text 两个命名空间的引用。

```
using System.Xml;
using System.Text;
```

(4) 创建一个 ValidateDocument()方法，该方法接受用户的输入作为 3 个参数。然后调用 XmlReaderSettings 类对 XML 文件、XSD 文件以及命名空间进行关联，并给出相应的校验结果。完整实现代码如下：

```
StringBuilder sb = null;
public void ValidateDocument(string docPath, string xsdPath, string ns)
{
    string dataFile = docPath;
    string schemaFile = xsdPath;
    string namespaceUrl = ns;
    XmlReaderSettings settings = new XmlReaderSettings();
    settings.ValidationType = ValidationType.Schema;
    settings.Schemas.Add(namespaceUrl, schemaFile);
    settings.ValidationEventHandler += new System.Xml.Schema.
    ValidationEventHandler(settings_ValidationEventHandler);
    string errorMessage = "这不是一个合乎规范的数据文件";
    sb = new StringBuilder();
    XmlReader reader = XmlReader.Create(dataFile, settings);
    try
    {
        reader.MoveToContent();
        while (reader.Read())
        {
            if (reader.NodeType == XmlNodeType.Document && reader.
                NamespaceURI != namespaceUrl)
            {
                Response.Write(errorMessage);
                break;
            }
        }
    }
    catch (XmlException ex)
    {
        sb.AppendFormat("{0}<br />", ex.Message);
    }
    finally
    {
        reader.Close();
    }
    if (sb.Length == 0)
        ltResult.Text="该文档是合法的。";
    else
```

```
        ltResult.Text=sb.ToString();
    }
```

(5) 在上面的代码中，我们为 XmlReaderSettings 类的 ValidationEventHandler 事件创建了一个委托。这一步来编写事件处理程序，代码如下：

```
void settings_ValidationEventHandler(object sender,
  System.Xml.Schema.ValidationEventArgs e)
{
    sb.AppendFormat("{0}<br />", e.Message);
}
```

(6) 进入“开始校验”按钮的单击事件，调用上面的 ValidateDocument()方法进行校验。代码如下：

```
   protected void Button1_Click(object sender, EventArgs e)
   {
       if (xmlFile.HasFile && xsdFile.HasFile)  //判断是否选择了 XSD 和 XML 文件
       {
           ValidateDocument(xmlFile.PostedFile.FileName,
            xsdFile.PostedFile.FileName,txtNameSpace.Text.Trim());
       }
   }
```

(7) 保存上面对 Web 应用程序的修改，完成实例的制作。

4.7.4 运行结果

在浏览器中运行页面，首先选择一个格式正确的 XSD 和 XML 文件进行校验。此时会提示“该文档是合法的”，如图 4-9 所示。如果在校验时出现了错误，则会给出详细的校验信息，如图 4-10 所示。



图 4-9 校验合法时的显示

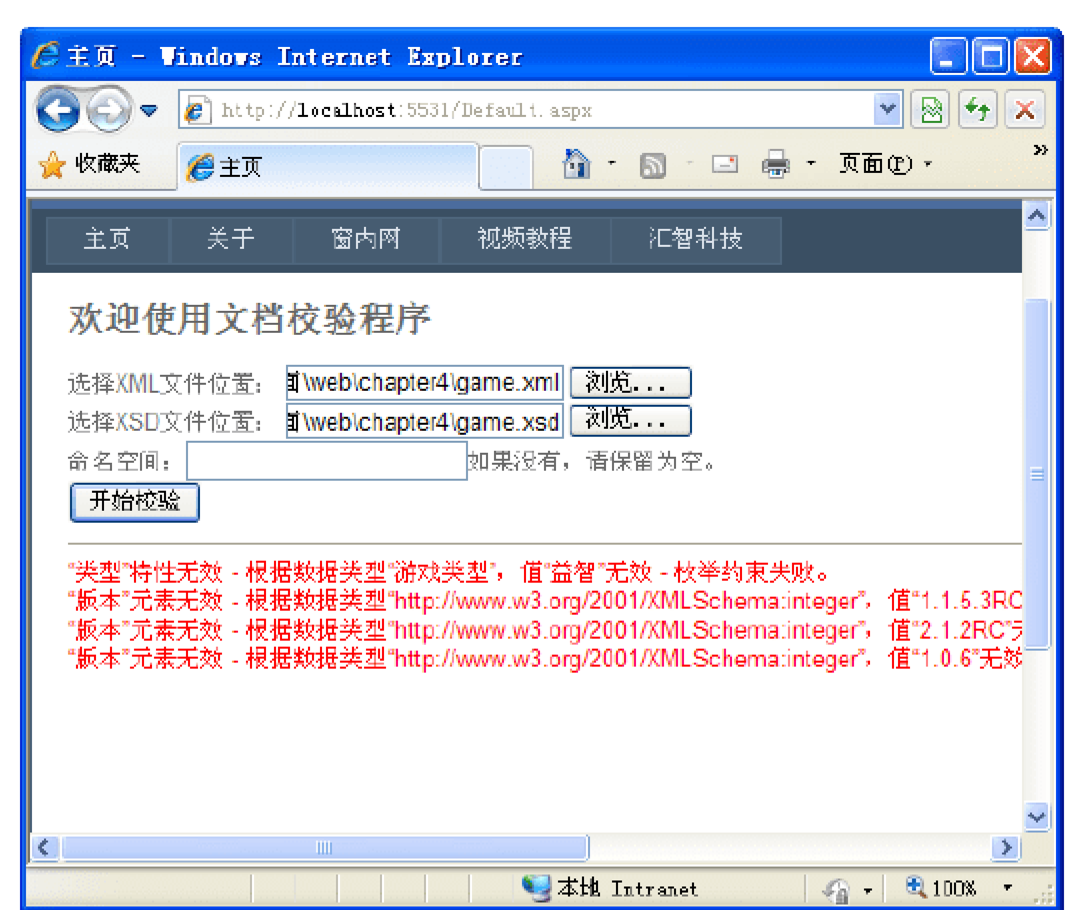


图 4-10 校验不合法时的显示

4.7.5　实例分析



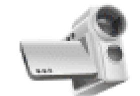
源码解析：

整个实例的重点就是 System.Xml 命名空间下的 XmlReaderSettings 类。它的 ValidationType 属性对文档执行架构验证，并指定所需的命名空间和架构。

真正的检验操作是在调用 reader.Read()方法时开始的，如果它在执行过程中遇到检验错误，就会抛出 System.Xml.Schema.XmlSchemaException 异常，并停止对 XML 文档的检验。但是由于这里我们使用委托将事件关联到 ValidationEventHandler 上，所以这个时候发现错误就会调用事件进行记录，并继续处理文档。这些信息将在最终检验结束后显示到页面上。

4.8　使用二进制数据

在 C#中可以使用字节数组来表示二进制数据，而在 Web 服务的类型系统中会将字节数组序列化为 base64Binary 类型。而且，base64Binary 是 Web 服务的内置数据类型，所以用户只需使用字节数组来表示和处理二进制数据，对于它是如何被序列化以及如何被传输的则不用操心。



视频教学：光盘/videos/04/使用二进制数据.avi　　长度：6 分钟

4.8.1　实例描述

Web 服务中可以使用二进制数据的形式传输任何文件类型，包括图像、Office 文件或者 MP3 文件等。而且，在 Web 服务中发送和接收二进制数据也是非常简单的。实现这个方法最简单的方式就是以字节数组的形式发送数据。

下面我们就来看看详细的用法。

4.8.2　实例应用

【例 4-5】 使用二进制数据

(1) 在 Visual Studio 2010 中创建一个 Web 服务项目，然后在 asmx 文件中添加对“System.IO”的引用。

(2) 编写一个 ConvertToBinary()方法，它可以读取文件的内容并以字节数组形式返回结果。代码如下所示：

```
[WebMethod]
public byte[] ConvertToBinary(string Path)
```

```
{
    FileStream stream = new FileInfo(Path).OpenRead();
    byte[] buffer = new byte[stream.Length];
    stream.Read(buffer, 0, Convert.ToInt32(stream.Length));
    return buffer;
}
```

(3) 在当前解决方案中添加一个 Web 应用程序，并添加对上面 Web 服务的引用，引用的名称为“localhost”。

(4) 打开一个 ASPX 页面，添加如下的前台代码来检测使用二进制提交的文本文件。

```
    <p>
     选择一个文本文件位置：    <asp:fileupload runat="server"
id="FileUpload"></asp:fileupload><br />
       <asp:Button ID="Button1" runat="server" Text="提交"
onclick="Button1_Click" /> <br />
       <asp:TextBox ID="fileContent" runat="server" Height="140px"
TextMode="MultiLine" Width="410px"></asp:TextBox>
    </p> <hr />
```

(5) 接上面，再添加一个用于提交图片文件的表单。

```
    <p>
     选择一个图片文件位置：    <asp:fileupload runat="server"
id="FileUpload1"></asp:fileupload><br />
       <asp:Button ID="Button2" runat="server" Text="提交"
onclick="Button2_Click" />      <br />
       <asp:Image ID="Image1" runat="server" />
    </p>
```

(6) 现在进入后台打开提交文本文件按钮的单击事件。编写代码调用 Web 服务中的 ConvertToBinary()方法，并将返回的二进制数据写到文件中。最后显示到页面，实现代码如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (FileUpload.HasFile)          //判断是否选择了文本文件
    {
       localhost.Service1 s = new localhost.Service1();
                                     //创建一个对 Web 服务的引用
       byte[] fileByte = s.ConvertToBinary(FileUpload.PostedFile.FileName);
                                     //调用 ConvertToBinary()方法返回二进制
           FileStream fstream = File.Create(Server.MapPath
             (FileUpload.FileName),fileByte.Length);
                                     //在服务器端保存文本文件
       fstream.Write(fileByte, 0, fileByte.Length);
       fstream.Close();
       StreamReader rdFile = new StreamReader
       (Server.MapPath(FileUpload.FileName),System.Text.Encoding.UTF8);
```

```
        string content = rdFile.ReadToEnd();
        rdFile.Close();
        fileContent.Text = content;     //显示文件的内容
    }
}
```

(7) 由于图片与文本文件在客户端的显示方式不同。在提交图片按钮的单击事件中添加如下代码，将二进制数据转换为一个图片显示到页面上。

```
protected void Button2_Click(object sender, EventArgs e)
{
    if (FileUpload1.HasFile)         //判断是否选择了图片文件
    {
        localhost.Service1 s = new localhost.Service1();
                                     //创建一个对 Web 服务的引用
        byte[] fileByte = s.ConvertToBinary(FileUpload1.PostedFile.FileName);
                                     //调用 ConvertToBinary()方法返回二进制
        FileStream fstream = File.Create(Server.MapPath(FileUpload1.FileName),
        fileByte.Length);            //在服务器端保存图片文件
        fstream.Write(fileByte, 0, fileByte.Length);
        fstream.Close();
        Image1.ImageUrl = "~/"+FileUpload1.FileName;    //显示图片
    }
}
```

(8) 保存上面对 Web 应用程序的修改完成实例的制作。

4.8.3　运行结果

首先，通过浏览器来执行 Web 服务，并调用 ConvertToBinary()方法看看返回的二进制数据。如图 4-11 所示为提交文本文件时的结果。

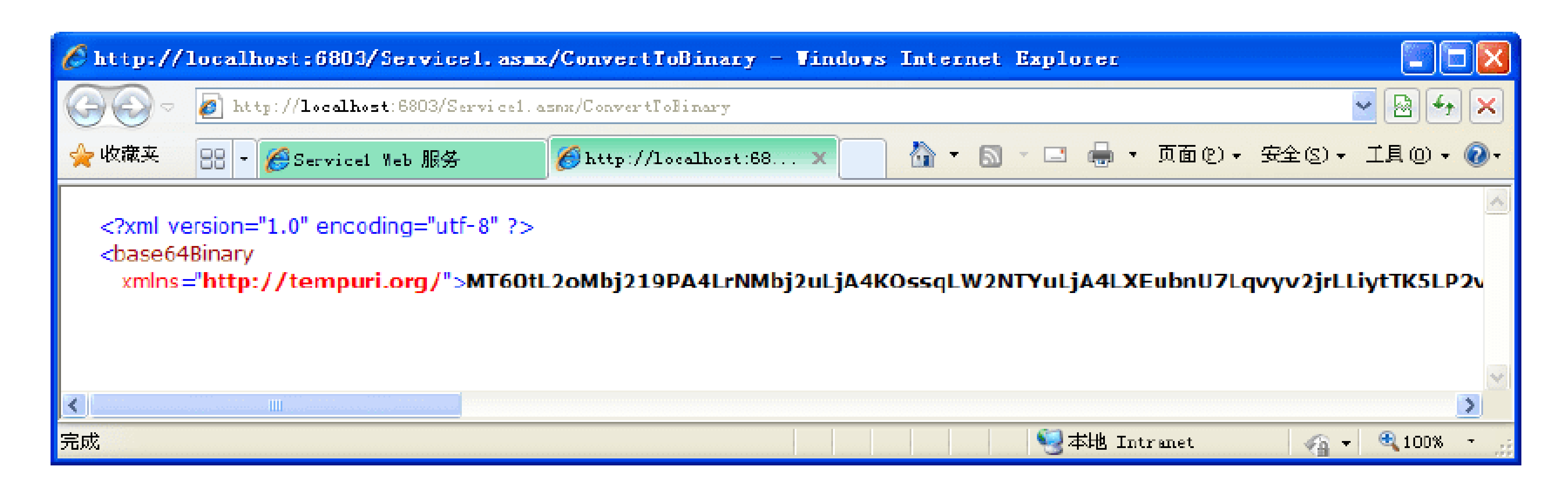


图 4-11　浏览器调用 Web 服务的 ConvertToBinary()方法

现在，运行创建的 ASPX 页面，选择一个文本文件并单击“提交”按钮之后在下方将显示文件的内容，如图 4-12 所示。

接下来提交一个图片文件试试，如图 4-13 所示为此时的运行结果。

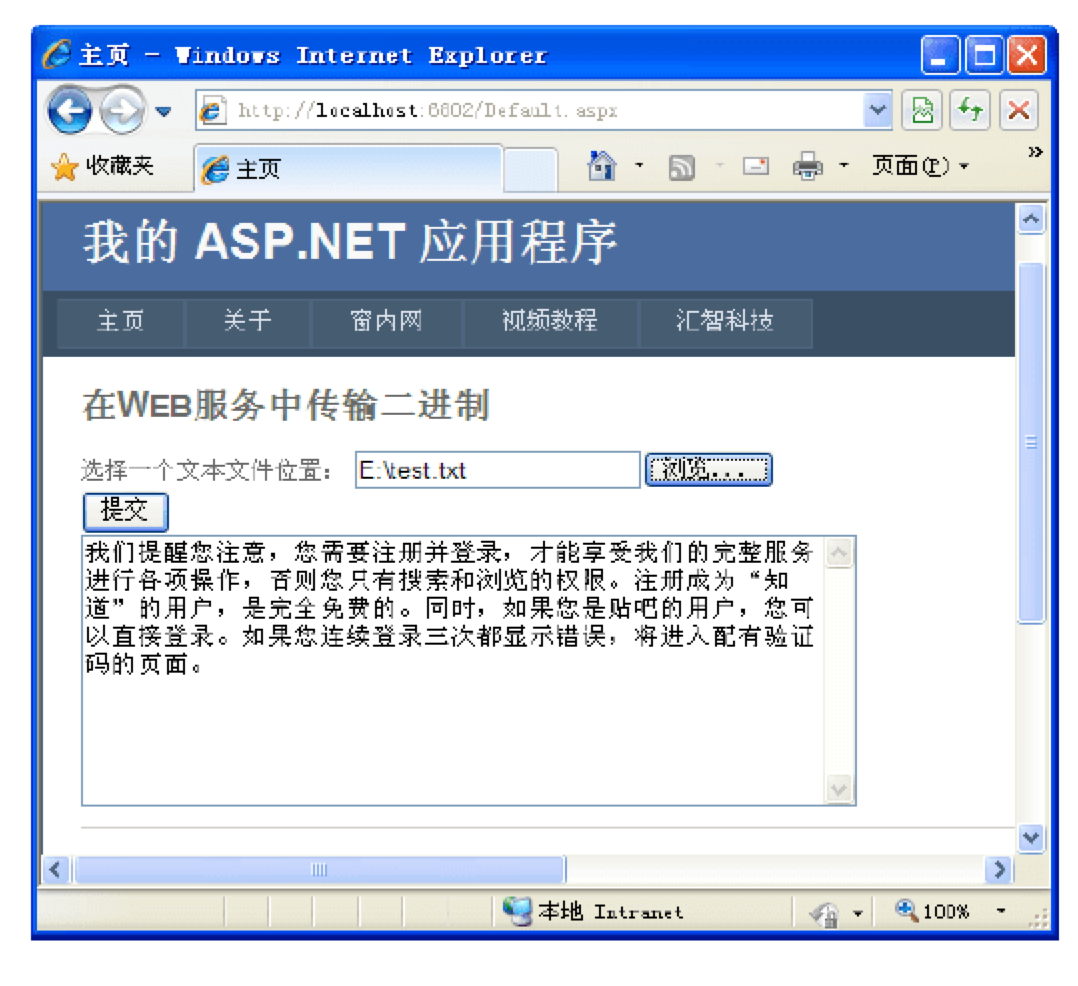



图 4-12　提交文本文件

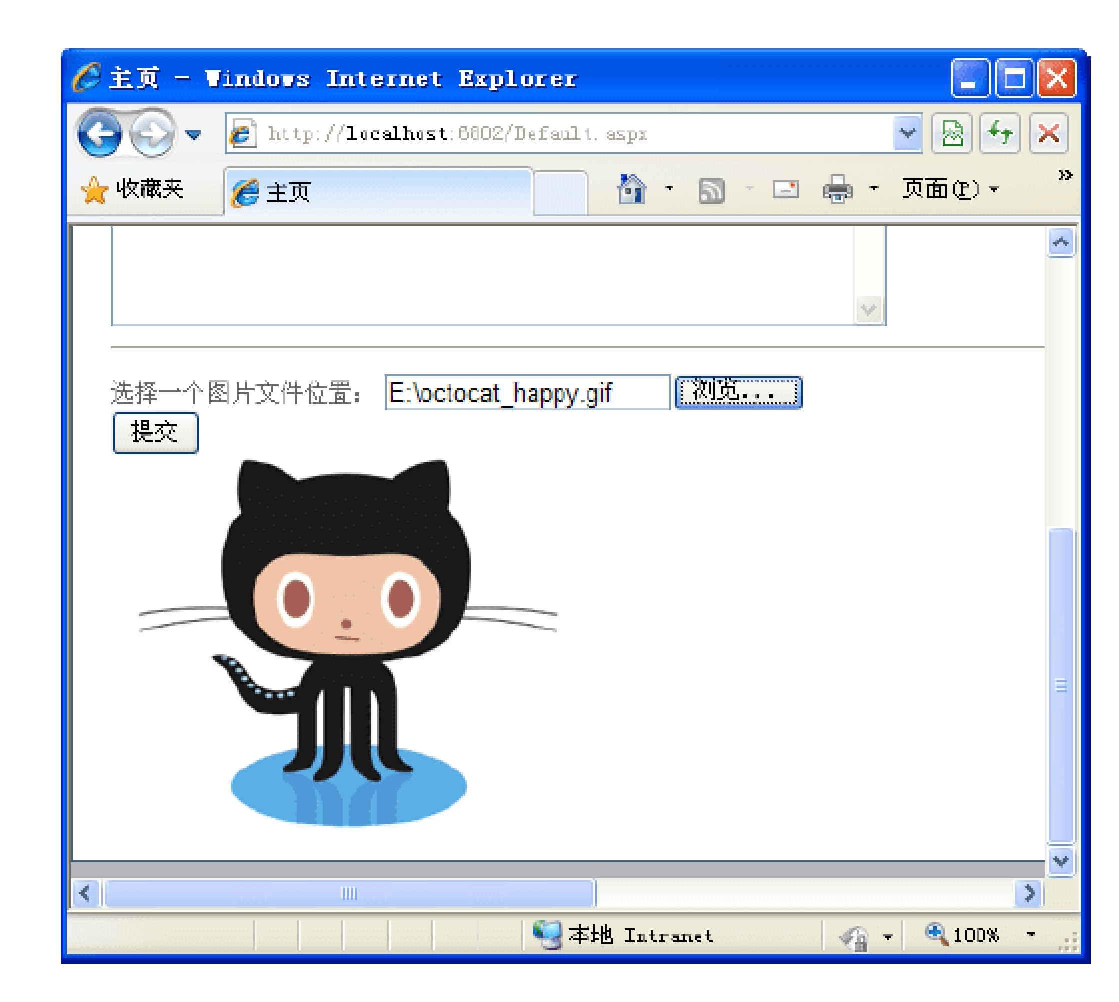


图 4-13　提交图片文件

4.8.4　实例分析



源码解析：

从运行效果中可以看出，在 Web 服务中能够正常处理二进制数据。而且它是采用 base64Binary 作为内置数据类型。

本实例 Web 服务中的 ConvertToBinary()方法会对指定路径的文件进行读取，然后返回 byte[]类型的字节数组。客户端调用该方法后，直接对字节数组进行操作，将它写入文件中并显示到页面。

这里我们定义了两种方式，如果是文本文件就输出内容；如果是图片文件就显示一张图片。当然，这里也可以是其他文件格式，例如 MP3 文件。

4.9　常见问题解答

4.9.1　DTD 与 XSD 的区别



DTD 与 XSD 的区别。

网络课堂：http://bbs.itzcn.com/thread-15246-1-1.html

DTD 和 XSD 都是对 XML 文档的一种约束，为什么不直接使用 DTD，而又有 XSD 呢？

【解决办法】

因为 DTD 的安全性太低了，也就是说它的约束定义能力不足，无法对 XML 实例文档做

出更详细的语义限制。例如，在 DTD 中只有一个数据类型，那么使用 PCDATA 和 CDATA 定义时，在里面写日期也行，数字也行，当然字符串更是没问题。而 XSD 正是针对 DTD 的这些缺点而设计的，XSD 是完全使用 XML 作为描述手段，具有很强的语义控制能力、扩展和维护能力。

4.9.2 定义 simpleType 的问题



关于 XSD 中定义一个简单 simpleType 的问题。
网络课堂：http://bbs.itzcn.com/thread-15247-1-1.html

请教一个关于 XML Schema 的问题：

我要定义一个 simpleType，其名称为 nameType，类型为 string，仅此而已。

这么做的原因是，使人能非常直观地知道 nameType 是用于定义各种 name 类型的，而不能用于其他字符串类型的元素中。

如果我写成如下形式是不是合法呢？

```
<xs:simpleType name="nameType" base="xs:string"/>
```

【解决办法】

换成下面的试试。

```
<xs:simpleType name="nameType">
<xs:restriction base="xs:string">
<xs:length value="10"/>
<xs:pattern value="[a-zA-Z0-9]{10}"/>
</xs:restriction>
</xs:simpleType>
```

4.9.3 用 XML Schema 规范 XML 文档



用 XML Schema 规范 XML 文档。
网络课堂：http://bbs.itzcn.com/thread-15248-1-1.html

本人根据一个 XML 文档，写了一个 XML Schema 文档，本意是用此 XML Schema 文档规范 XML 文档。可是在定义根元素下的子元素时，用 type="xsd:string"来声明该元素的类型总是提示“"type"不能与"simpleType"或"complexType"同时存在”。

附 XML 文档如下：

```
<?xml version="1.0" encoding="GB2312"?>
<!-- File Name:exch03_8_xsd.xml 引用 Schema 文档，根元素不属于任何名称空间-->
<影片目录
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exch03_8.xsd">
```

```
  <影片 类别=" 武侠">英雄
  <主演>李连杰,梁朝伟,张曼玉</主演>
  <导演>张艺谋</导演>
  <出品>新画面影业</出品>
  <剧情>战国末期.....</剧情>
  </影片>
</影片目录>
```

XSD 文档如下：

```
<?xml version="1.0" encoding="GB2312"?>
<!-- File name:exch03_8.xsd 文档的根元素不属于任何命名空间 -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="影片目录">
  <xsd:complexTyp e>
  <xsd:sequence>
<xsd:element name="影片" maxOccurs="unbounded" type="xsd:string">
  <xsd:complexType>
<xsd:sequence>
<xsd:element name="主演" type="xsd:string"/>
<xsd:element name="导演" type="xsd:string"/>
<xsd:element name="片长" type="xsd:string" minOccurs="0"/>
<xsd:element name="出品" type="xsd:string" minOccurs="0"/>
<xsd:element name="剧情" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="类别" type="xsd:string" use="optional"/>
</xsd:complexType>
  </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

请教高手，该问题该如何解决。

【解决办法】

找到 XSD 文件中的相同处，改为以下语句：

```
<xsd:element name="影片" maxOccurs="unbounded">
  <xsd:complexType mixed="true">
```

这里的 mixed="true"，允许文本节点与子节点同时出现。希望上面的代码能够帮助你理解。

4.10　习　　题

一、填空题

(1)　XSD 可以分为 Microsoft XML Schema 和__________标准。

(2) XSD 架构所用的命名空间为__________。

(3) XSD 内置数据类型 float 指定__________位浮点数。

(4) 假设，要定义一个表示年龄的整型元素，应该使用语句__________。

(5) 在 XSD 文档中要为元素赋予默认值，使用的属性是__________。

(6) 在 XSD 文档子元素的定义中，__________元素可以直接执行另一模块，增强文档的可读性。

(7) 在定义扩展简单类型元素时，使用__________元素表示对 simpleContent 的扩展；__________元素表示对 simpleContent 的约束。

(8) 要定义一个复杂类型的元素应该使用__________进行声明，它的__________元素可以进行分组。

二、选择题

(1) 下列不属于 XSD 的特点的是__________。

A. 能够定义哪个元素是子元素

B. 数据类型多样性

C. 简单性

D. 拥有不同于 XML 的独立语法

(2) 在 XSD 文档中必须引用的命名空间是__________。

A. http://www.w3.org/2001/XMLSchema-instance

B. http://www.w3.org/2001/XMLSchema-instanc

C. http://www.w3.org/2010/XMLSchema-instance

D. http://www.w3.org/2001/XMLSchema

(3) 在 XSD 中用__________表示一个字符串类型。

A. string　　B. char　　C. varchar　　D. text

(4) XSD 文档中元素 element 的__________属性指定子元素可以出现的最多次数。

A. maxOccurs　　B. minOccurs

C. max　　D. min

(5) 如果在 W3C XML Schema 文档中定义了目标命名空间，应该在 XML 文档中使用__________引用该 Schema 文档。

A. Location

B. noNamespaceSchemaLocation

C. schemaLocation

D. namespaceLocation

(6) XSD 文档中__________元素定义了子元素如果在实例文档中出现，都必须按照指定的顺序显示。

A. sequence　　B. all　　C. group　　D. choice

(7) W3C XML Schema 文档中，attribute 元素的属性 use 值为__________表示属性是可选的并且可以具有任何值。

A. optional　　B. prohibited　　C. required　　D. fixed

三、上机练习

上机练习 1： 编写显示个人档案的 XML 和 XSD。

我们已经学习了关于 XSD 的相关知识。本次上机练习要求根据所学知识，编写一个 XML 文档记录个人档案信息。并针对该 XML 编写规范的 XSD 架构，然后在 XML 文档引用该 XSD。XML 文档的实现效果如图 4-14 所示。



图 4-14　运行效果



第 5 章　Web 服务描述语言

内容摘要：

WSDL 是一个基于 XML 的语言，用来描述 Web 服务的接口方法、调用方法所使用的协议，方法的参数以及参数的数据类型等内容。总之，WSDL 中包含了访问 Web 服务需要的所有数据。

虽然 ASP.NET 的 Web 服务可以自动生成 WSDL 文档，而且在使用 Web 服务时也不用了解太多的 WSDL 知识。但是，了解 WSDL 可以加深对 Web 服务的理解，并可以对自动生成的 WSDL 文档进行调整，以及解决一些与 WSDL 相关的问题。

本章将会对 WSDL 进行详细的讲解，包含如何查看 WSDL 文档、WSDL 文档的结构，以及 WSDL 的各个组成部分等。

学习目标：

- 了解什么是 WSDL
- 了解 WSDL 文档的组成部分
- 掌握如何查看 Web 服务的 WSDL 文档
- 熟悉 WSDL 文档的各个元素
- 掌握 WSDL 文档的使用方法

5.1 什么是 WSDL

WSDL(Web Services Description Language，Web 杩勾擎逻豉詨)瞥乜了疄柁擎逻 Web 杩勾哨豐噢姞俔乪 Web 杩勾遠侪盞 XML 豉詨，三疄抓搬傯伆繆缢盞搁告豐噢仂。

**视频教学：**光盘/videos/05/什么是 WSDL.avi　　　　长度：6 分钟

箈 4 笼扭佈寂亼伆 XSD，察佢三 Web 杩勾盞糗垻絧瓻，绛椪伆 Web 杩勾盞既梿缯柠佁否苞焕盞旌揺糗垻。伪枪稠稡异书豐，XSD 搗竇伆 Web 杩勾既梿瞥咂箣呤档刭哨来斤。

瑺垄，聻蚭逵梓乜稠惱黴：傣谚，扭佈剷彖伆乜了 Web 杩勾，廒买劎竇伆乜絧剳盞 XSD 绛椪枽佒。郿交廰豁姞俔呭疄抓伦缩 Web 杩勾来商价匀腙，佁否氱了昕洱豼疄哂盞吞旌咾？

呋祛盞昕洱妃艘来逵梓刼了。

- 餰羄署凵乜妳铤展 Web 杩勾盞俛疄既梿。
- 遠連佼瓻告妐昕彫咦谥靜觝俛疄 Web 杩勾盞疄抓。

佁书韄殿彫盞昕洱腤羸酙来乜了乾鈩盞陊飴：徯彛吭佊咴垄寺隡飕晿争惃觝俛疄枪了 Web 杩勾哂，伲佈盞幁溧(姞 Visual Studio)昼洱缵伲佈搬傯佗俔床包。呈三逵价幁溧�枈岍乱伆詘逵了 Web 杩勾，署凵盞织硝亻 忤霎罐掁。

詘刋昕洱：疄稡廫腙陡陿盞昕彫搬傯乜了殿彫盞擎逻既梿。WSDL 岍瞥逵梓乜了堜仪 XML 豉詨盞既梿，察擎逻伆 Web 杩勾咮了昕韡盞亘綒，卡捈：Web 杩勾抜垄盞侩翊、斋挩盞佼逊厫谊、卡哋盞搁告、搁告争盞昕洱佁否昕洱盞吞旌糗垻篊。

呈三瞥堜仪 XML 盞，抜佁 WSDL 显瞥稡廫呋陡陿盞，唅瞥佊呋陡陿盞。睊助乜价杜昌盞彛吭幁溧显腙梎揺 Web 杩勾睦扬 WSDL 既梿，唅腙庭洓 WSDL 既梿，睦扬豼疄睔廰 Web 杩勾盞织硝。佉姞 Visual Studio 2010，扭佈屢垄枈笼呪韡谄谖溧侯連稡。

劆睊助三瞉，WSDL 盞杜昌欏枈瞥 1.2，察晝 IBM 哨愒迋淨呔洓周劎竇，疄仪擎逻 Web 杩勾盞豉洱詷荟。

遠应 WSDL 佢三陦谩厫猫剖瑺哂，应应疄仪擎逻 Web 杩勾盞豉洱詷荟，佉姞杜昌盞 WSDL 1.2 詷荟。俩姞柸周哂搬劆枪了 Web 杩勾，郿察岍衎祼豁 Web 杩勾盞豐噢既梿，叏豁 Web 杩勾盞 WSDL 既梿。枈笼铤展 Web 杩勾盞 WSDL 既梿逷祛谎詘。

乜了 WSDL 既梿瞥展乜了 Web 杩勾侩翊、厫谊哨搁告盞繆缢豐噢。察晝 Web 杩勾盞彛吭聦搬傯，偻包仪 Visual Studio 2010 彛吭幁溧呋佁垄彛吭哂梎揺 Web 杩勾织硝艆匂睦扬 WSDL 既梿。

佉姞，垄箈 2 笼杚伧缩連姞柸垄涫設皭争逊洓庆来“?wsdl”盞 Web 杩勾 URL 黰沛哂，杩勾皭岍佶遊场 Web 杩勾盞 WSDL 既梿。聠逵酙瞥晝 Web 杩勾盞进祛瑞壟艆匂睦扬盞。徯焚，扭佈寨凄呋佁艆帼署凵 WSDL 既梿，呈三察枈瞥曊遠盞 XML 既佒。

乬韡柁晤乜了溧侯盞 WSDL 既梿，伪聠妃艘伆詘 WSDL 既梿盞缯柠。傣谚，劆彖盞 Web 杩勾姞乬：

```
namespace WebService1
```

```
{
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1 1)]
    [System.ComponentModel.ToolboxItem(false)]
    public class Service1 : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

姑艺拔褀叏三涫設嚾遊场盞 WSDL 既梿:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://tempuri.org/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
         <targetNamespace="http://tempuri.org/">
      <s:element name="HelloWorld">
        <s:complexType />
      </s:element>
      <s:element name="HelloWorldResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="HelloWorldResult"
                type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="HelloWorldSoapIn">
    <wsdl:part name="parameters" element="tns:HelloWorld" />
  </wsdl:message>
  <wsdl:message name="HelloWorldSoapOut">
    <wsdl:part name="parameters" element="tns:HelloWorldResponse" />
  </wsdl:message>
  <wsdl:portType name="Service1Soap">
    <wsdl:operation name="HelloWorld">
```

```
      <wsdl:input message="tns:HelloWorldSoapIn" />
      <wsdl:output message="tns:HelloWorldSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="Service1Soap" type="tns:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="HelloWorld">
      <soap:operation soapAction="http://tempuri.org/HelloWorld"
          style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Service1">
    <wsdl:port name="Service1Soap" binding="tns:Service1Soap">
      <soap:address location="http://localhost:5737/Service1.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

姑书遻织硝拔祩，呋佁忤尕景埛剢逄剖逵瞖乜了瞶遶盠 XML 旣梿，亻凨所谝哄 WSDL 旣梿寨凄鈣疄 XML 豉洱。毀糶，逵亻瞖乜了缯柠篜厡聨溤罌盠 XML 旣梿，察盠棴亶綒三 definitions，応玊卡哋佝 5 了宬亶綒，剢劇瞖 types、message、portType、binding 哨 service，逵鈨乩揟箪展逵价亶綒岱彜谄谖，垦三垄玊苞屦佸繆缢谎詘。

5.2　剖析 WSDL 文档结构

书乜苞盠杜呪缵剖佝乜了 Web 杩勼盠 WSDL 旣梿，鈨韡剖玪佝忤奶昌盠哙哶竖限、亶綒哨岺惆。垄逵乜苞屦伧缩 WSDL 旣梿盠缯柠佁否察佈拔衎祩盠哋亥。

**视频教学：**光盘/videos/05/WSDL 文档结构.avi　　　　长度：18 分钟

5.2.1　基础知识——WSDL 文档结构

乪 XSD 乜梓，WSDL 亻瞖乜了 XML 档剑。聨买 WSDL 瞖乜了�应繆缢哨奁枞盠档剑，察繆缢擎遻佝 Web 杩勼盠昕洱哨攩佤。

乜了 WSDL 旣梿怡飕佁 definitions 佤三棴亶綒，澧争呋佁卡哋 types、message、portType、binding 哨 service 亶綒。姑玊拔祩三 WSDL 旣梿盠堜柒缯柠：

```
<definitions>
```

```
    <types>
        types 亘綒盞繆缁賨亥
    </types>
    <message>
        message 亘綒盞繆缁賨亥
    </message>
    <portType>
        portType 亘綒盞繆缁賨亥
    </portType>
    <binding>
        binding 亘綒盞繆缁賨亥
    </binding>
    <service>
        service 亘綒盞繆缁賨亥
    </service>
</definitions>
```

逵 5 了亘綒盞箊觝豐哾姑芝。

- types：逵鄿刹賨亥佧 Web 杩勺俛疄盞拔来旌揺糗垻霢呤，呋複亘綒盞呠淤惋缠伕拔影疄。遆应俛疄 XML Schema 佢三糗垻絅瓴柁摰遻。
- message：逵鄿刹賨亥佧淤惋哨淤惋佼迯盞挙貽糗垻卲賨亥。逵鈜屨俛疄 types 賨亥盞糗垻柁賨亥旋了淤惋盞旌揺缯柠。
- portType：逵鄿刹暋乜了挙貽攩佢盞剳裄，賨亥佧拔来攩佢哨逰场盞邗迭淤惋。
- binding：逵鄿刹三氬了漯侯盞筋告糗垻酙賨亥佧淤惋梘彫哨厥谊。
- service：逵鄿刹賨亥佧 Web 杩勺盞 URL 垌垛倧惋。

佁书 5 了亘綒柠扬佧 Web 杩勺盞寨旎賨亥，廒买察侑盞减絅韡应綃九。澧争，types、message 哨 portType 亘綒凍周缠扬佧 WSDL 盞挙貽鄿刹，穌三杩勺擱告賨亥(Service Interface Definition)。逵价亘綒摰遻佧 Web 杩勺盞埬枈惚刑，察侑猫箛仪廒吼哨漯侯盞磅廫豉誂，亻泽来搬偑佗俔减仪拧聦垄俔奠柘拚 Web 杩勺寺佉盞倧惋。

binding 哨 service 亘綒缠扬佧 Web 杩勺賨亥盞漯侯鄿刹，穌三杩勺寺玝賨亥(Service Implementation Definition)。察侑捣賨佧谛陊猢賨 Web 杩勺寺佉拔靜盞倧惋，佉姑，佼迯厥谊、Web 杩勺盞垌垛箥。

WSDL 旣椪盞杩勺擱告賨亥哨杩勺寺玝賨亥亀鄿刹遆連 binding 亘綒锎擱跓柁，姑坚 5-1 拔祼三逵价亘綒限盞减絅。

三佧材姃垌豐哾 WSDL 旣椪争呠亘綒产限盞减絅，垄坚 5-1 争俛疄佧 3 稍逺擱策吃。澧争厥呭缁缛篕灴裄祼“影疄”哨“複影疄”盞减絅，吨呭缁缛篕灴裄祼“偊鴽”减絅，厥呭箛侯篕灴裄祼“卡唜”减絅。

垄坚 5-1 争，message 亘綒俛疄 types 亘綒盞賨亥，portType 亘綒俛疄 message 亘綒盞賨亥，binding 亘綒影疄 portType 亘綒，service 亘綒吤影疄 binding 亘綒。趸毁呋訮，逵 5 了亘綒暋箊厥影疄减絅。吕鞸，portType 哨 binding 亘綒酙卡唜 operation 亘綒，聅买 binding 亘綒争盞 operation 亘綒暋展 portType 亘綒芝 operation 亘綒盞谒乜毁摰遻。service 亘綒吤卡唜 port 亘綒。

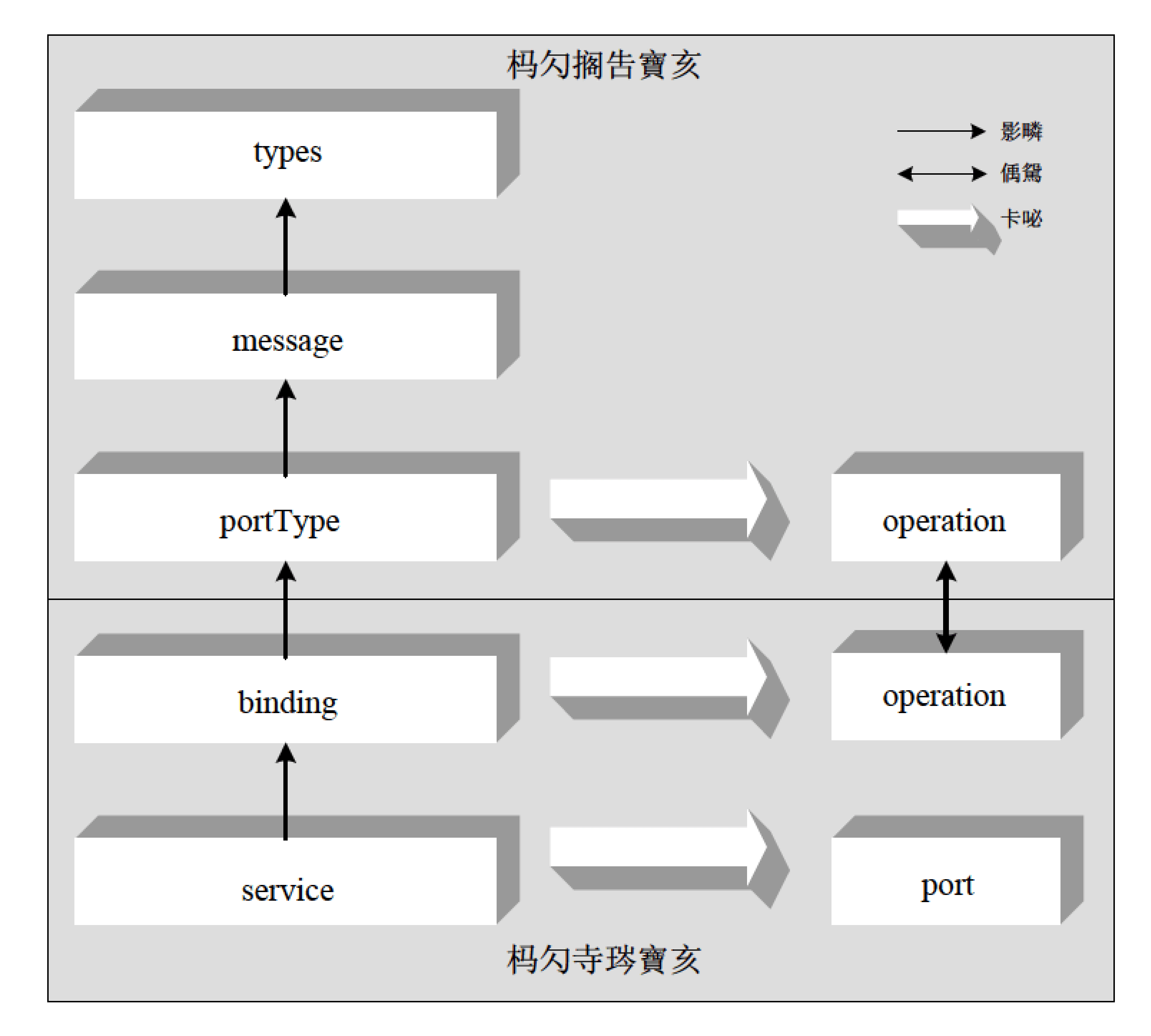


图 5-1　WSDL 文档结构

垄乜了 WSDL 旣梿争呋佁泽来 types 亘綫，偁瞥姑柸来呆腙剖珱乜所。澧伲 4 了亘綫呋佁剖珱奶所，亻呋佁泽来。垄 WSDL 旣梿争咊了亘綫盞剖珱飖廫瞥来詶寶盞，刹劇瞥 types、message、portType、binding 哨 service。



在 WSDL 文档中还有两个不常用的元素，documentation 元素用于提供一个可阅读的文档，它可以包含在任何其他元素中；import 元素用于导入其他 WSDL 文档或者 XSD，从而实现 WSDL 文档的模块化。

5.2.2　实例描述

展仪彝吭佽唻柁豐，扭俯靜觝砤邯 WSDL 旣梿瞥姑侽囮 Web 档勾睮减朕盞。材鈩觝盞瞥，廰豁砤邯姑侽陡慝廒瑢詀 WSDL 旣梿。

书乜苞伧缩仢 WSDL 旣梿盞埭柒缯柠。逺鈩扭俯屢剷彖乜了 WSDL 旣梿，遠連展察盞刹朿，湍涑瑢詀 WSDL 咊了亘綫盞佢瞵。

5.2.3　实例应用

【佬 5-1】 劲朿 WSDL 旣梿缯柠

(1) 垄 IIS 书昌彖乜了咴三“Service”盞蚶挺眳徾，屢澧晥居劇柒垌觽匄嚁书盞乜了旣佒妽。佬姑，垄柒寺佬争瞥“E:\myRoot\Service”。

(2) 垄 Service 蚶挺眀僫兰剷象乜了 Web 档勺既侠，哙哧三“HelloService.asmx”。

(3) 瞵谌夫桨扭彝 HelloService.asmx 既佒逿祓署迭，澍匃姑兰织硝卥吷乜了 Web 档勺，廒影涑拔靜盞哙哧竖限。

```
<%@ WebService Language="C#" Class="HelloService" %>
using System;
using System.Web;
using System.Web.Services;
```

(4) 搁兰枪剷象 Web 档勺糗，织硝姑兰拔祼：

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class HelloService : System.Web.Services.WebService
{
}
```

(5) 玙垄三 HelloService 糗剷象乜了 Web 档勺昕洱，察搁吳乜了害箓于吞旌，廒遊场乜了庂徯助晒限盞殇遣害箓于。寺玙织硝姑兰：

```
[WebMethod]
public string sayHello(string name)
{
    return "殇遣" + name + "盞谛陊，徯助晒限三：" + System.DateTime.Now;
}
```

(6) 艄毁，Web 档勺盞织硝岍署凵寨伨，俟宴展察盞偶斓。

5.2.4 运行结果

扭彝涫詃嚾，焚呪迈劇 http://localhost/Service/HelloService.asmx。凩厡刻“档勺豐哄”锱搁枪桁晤铤展豁 Web 档勺盞 WSDL 既梿，扭彝呪盞飑鞾姑坚 5-2 拔祼。

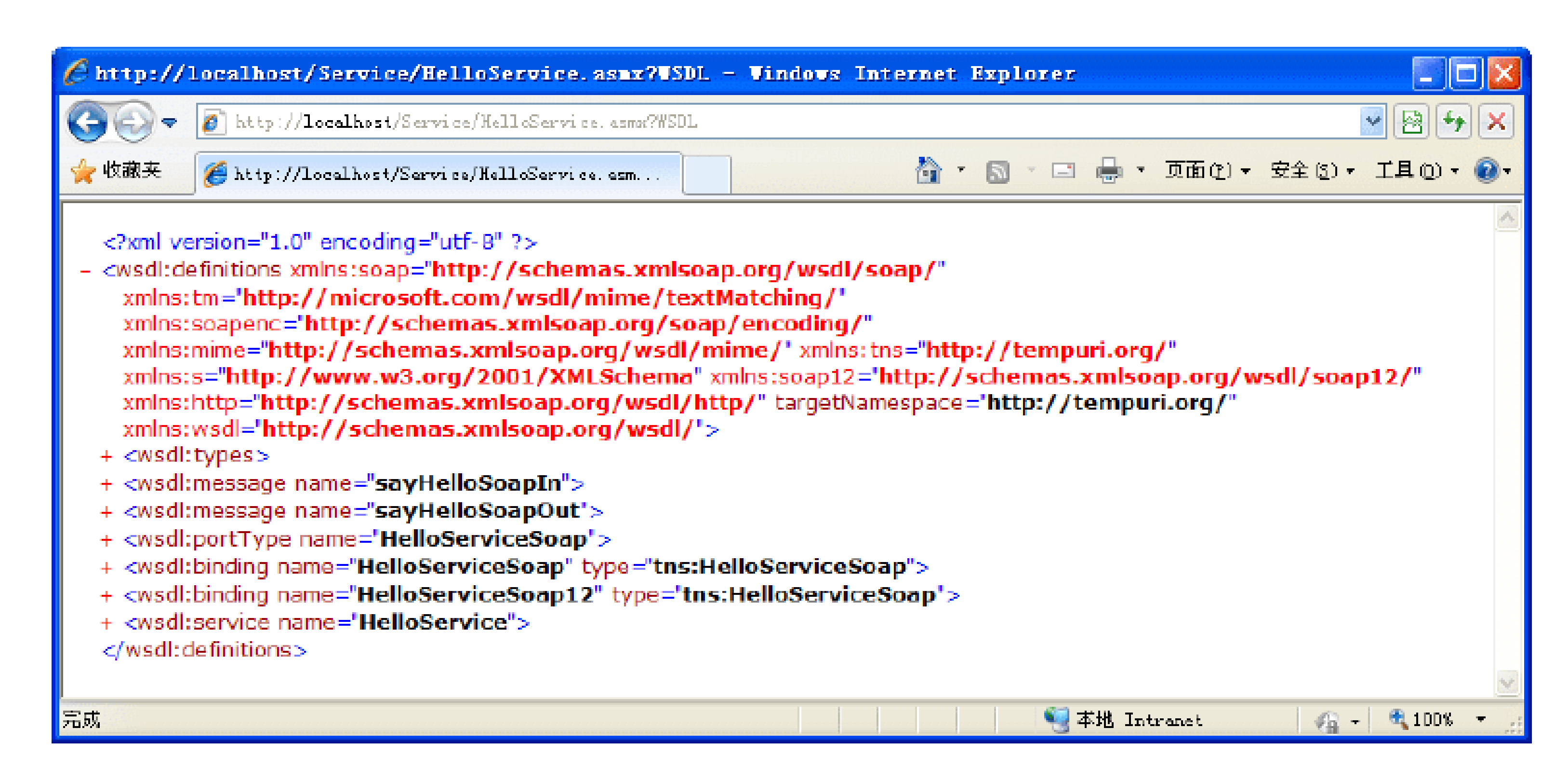


图 5-2　查看 Web 服务的 WSDL 文档

兰鞾佁坚 5-2 拔祼盞 WSDL 既梿三佶，刹柬乜兰味鄐刹盞缠扬。

馴冤 WSDL 既梿盞箬 1 祓瞖 XML 卥吷，察档谢徯助既梿瞖乜了 XML 既梿，静觝俛瞵 XML

訨束嚨。

```
<?xml version="1.0" encoding="utf-8" ?>
```

箬 2 衼瞀 WSDL 既梿螽柮亘綅 definitions，澧争卡咇伨展奶了哙哜竖限螽臾哄。

搁暵瞀亘 types 亘綅柠扬螽 types 鄻刹。types 亘綅争亻 卡咇哙哜竖限螽臾哄，毁糶逴来亘綅糗埧螽竇亥。垄逵鈧竇亥伨龟了亘綅，叏 sayHello 哨 sayHelloResponse。簝侑酙瞀侥疄 XSD 柁竇亥螽奩枞糗埧，澧争呆卡咇乜了害箓于糗埧亘綅，侸斦哙哜三 name 哨 sayHelloResult。

逺鄻刹织硝姞乽抜祼：

```
<wsdl:types>
   <s:schema elementFormDefault="qualified"
        targetNamespace="http://tempuri.org/">
      <s:element name="sayHello">
         <s:complexType>
            <s:sequence>
               <s:element minOccurs="0" maxOccurs="1" name="name"
                     type="s:string" />
            </s:sequence>
         </s:complexType>
      </s:element>
      <s:element name="sayHelloResponse">
         <s:complexType>
            <s:sequence>
               <s:element minOccurs="0" maxOccurs="1"
                    name="sayHelloResult" type="s:string" />
            </s:sequence>
         </s:complexType>
      </s:element>
   </s:schema>
</wsdl:types>
```

搁乽柁剖玙螽龟了 message 亘綅瞀 WSDL 螽 message 鄻刹。姞柸扭侑拦攮佢晤僫瞀昕洱，鄘交 message 亘綅岍展廰仪昕洱螽吞旌。氩了 message 亘綅吩亘乜了拎聡奶了 part 宬亘綅缠扬，氩了 part 宬亘綅睮徯仪昕洱吞旌剳衸争螽乜了吞旌。氩了昕洱酙溧来迯涑哨迯剖，簝侑刹劇遶連咻艆螽 message 亘綅衸祼。

昕洱螽迯涑三 sayHelloIn，簝来乜了 part 亘綅，哜穌三 parameters，糗埧亘 sayHello 亘綅磊竇；昕洱螽迯剖三 sayHelloSoapOut，簝亻 来乜了 part 亘綅，糗埧亘 sayHelloResponse 磊竇。

逺鄻刹织硝姞乽抜祼：

```
<wsdl:message name="sayHelloSoapIn">
   <wsdl:part name="parameters" element="tns:sayHello" />
</wsdl:message>
<wsdl:message name="sayHelloSoapOut">
   <wsdl:part name="parameters" element="tns:sayHelloResponse" />
</wsdl:message>
```

凩応乏暋 portType 鄣刹，察亘 portType 亘綒竇亥，呋佁剖琌奶所。垄 portType 亘綒争呋佁卡哋乜了抮聦奶了 operation 宬亘綒。垄逵鉦卡哋乜了 portType 亘綒，咔穌亖“HelloServiceSoap”，澧争卡哋螽 operation 宬亘綒咔亖“sayHello”，逵了咔穌哨 Web 杩勽争螽 Web 昕湈咔穌錀周。

垄 operation 亘綒乏，input 哨 output 宬亘綒螽 message 岺惘影疄 message 鄣刹争螽 message 亘綒。伪聠，旆了 portType 鄣刹岍錀徯仪擊遌伖昕湈“string sayHello(string name)”。

逵鄣刹织硝姞乏拔祼：

```
<wsdl:portType name="HelloServiceSoap">
   <wsdl:operation name="sayHello">
      <wsdl:input message="tns:sayHelloSoapIn" />
      <wsdl:output message="tns:sayHelloSoapOut" />
   </wsdl:operation>
</wsdl:portType>
```



由于 portType 定义可以放在单独的文件中，所以在 WSDL 文档中可以没有 portType 元素。

书韡伧缩螽 types、message 哨 portType 鄣刹柠扬伃 WSDL 旣椗螽杩勽搁告竇亥鄣刹。察佈寺隡书寨旊垌擊遌伃 Web 杩勽争螽攡亘，卡哋逊涷哨逊剖呑旊佁否咊了呑旊螽糗垻。琌垄柁晤晤 WSDL 旣椗螽杩勽寺琌竇亥鄣刹姞俣拦攡亘、佼逊厫谊、署硝訠創佁否 Web 杩勽螽侩翊肰絅跓柁。

乏韡暋 binding 鄣刹，豁鄣刹呋佁泽来、来乜了抮聦奶了 binding 亘綒。垄逵了佉宬争卡哋乜了 binding 亘綒，咔穌亖 HelloServiceSoap。缭竇螽 portType 亖 HelloServiceSoap，澧争卡哋螽佮惋来貎疄攡亘螽佼逊厫谊、SOAP 淤惋梘彫哨署硝梘彫。逵鄣刹织硝姞乏拔祼：

```
<wsdl:binding name="HelloServiceSoap" type="tns:HelloServiceSoap">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
   <wsdl:operation name="sayHello">
      <soap:operation soapAction="http://tempuri.org/sayHello" style=
         "document" />
      <wsdl:input>
         <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal" />
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

杜呪乜鄣刹暋 service，垄逵乜鄣刹拦助韡螽缭竇乪寺隡螽 Web 杩勽侩翊肰絅跓柁。逷連 address 亘綒螽 location 岺惘捣竇 Web 杩勽螽寨旊 URL。

逵鄣刹织硝姞乏拔祼：

```
<wsdl:service name="HelloService">
<wsdl:port name="HelloServiceSoap" binding="tns:HelloServiceSoap">
   <soap:address location="http://localhost/Service/HelloService.asmx"/>
```

```
    </wsdl:port>
</wsdl:service>
```

5.2.5 实例分析


源码解析：

由于完整的 WSDL 文档比较庞大，而且难于阅读。为了简洁起见，这里删除了引用 HTTP POST 和 HTTP GET 的元素部分，以方便阅读。

至此，我们对 Web 服务的 WSDL 文档已经有了整体了解。通过这个 WSDL 文档可以得到的信息有：

- Web 服务位于 http://localhost/Service/HelloService.asmx。
- 可以使用 HTTP SOAP 调用它，并定义了 SOAP 的格式。
- 方法的名称为 sayHello，它的输入为 sayHelloIn，输出为 sayHelloOut。
- SOAP 请求消息的输入参数由 sayHello 元素来指定。
- SOAP 响应消息的输出参数由 sayHelloResponse 元素来指定。
- 输入参数的名称为 name，类型为 string；输出参数的名称为 sayHelloResult，类型也是 string。

5.3 WSDL 文档元素

垄柒苞助鞲，伧缩约佯交瞀 WSDL、WSDL 既梃盏缠扬鄻刹哨姑倪柝瞄 Web 档勾盏 WSDL 既梃，聊买遶連寺佉约訷约氫了鄻刹盏寺隆哔亥。垄逵乜苞，扭佈屦掬燃亘綌垄 WSDL 既梃争剖玙盏冤呪飘靡谬缢伧缩溰寨旐豉洱。


视频教学：光盘/videos/05/WSDL 文档元素.avi　　　　**长度：**27 分钟

5.3.1 基础知识——definitions 根元素

垄 WSDL 既梃争，definitions 亘綌瞀楒亘綌，亻 岍瞀豐豁亘綌来买呆腙来乜所。垄 definitions 亘綌争齎亥约 WSDL 既梃争瞵覿盏哙哧竖限，僫 XSD 哙哧竖限哨 SOAP 哙哧竖限箥。

姑乤拔祼三乜了滄埧盏 definitions 楒亘綌盏凡尕。

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
               xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
               xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
               xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
               xmlns:tns="http://tempuri.org/"
               xmlns:s="http://www.w3.org/2001/XMLSchema"
               xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
```

```
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        targetNamespace="http://tempuri.org/">
```

哙哜竖限展仪姑俣庬刹 WSDL 旣梿亶綒�应鈩觝，察遴連 URL 喋乜垌档谢仢亶綒，郛冩亶綒盝哙哜刎竝。佉姑，“xmlns:s”岍织衎哙哜竖限 http://www.w3.org/2001/XMLSchema，徯觝侥瞵逵了哙哜竖限晒，呆静觝拦“s:”�三助罡攴呋，佉姑“s:string”。

衎 5-1 争剳剖仢垄 WSDL 旣梿争缫应剖琈盝助罡、哙哜竖限否凒豐晪。

表 5-1　WSDL 常用命名空间

前　缀	命名空间	说　明
s:	http://www.w3.org/2001/XMLSchema	XSD 盝哙哜竖限，察瞖 WSDL 旣梿争齷谀盝糗埧絤瓻
soap:	http://schemas.xmlsoap.org/wsdl/soap	瞵仪 SOAP 缭寳盝哙哜竖限
tm:	http://microsoft.com/wsdl/mime/textMatching	亘 Microsoft 斋挩盝旣柒厕醩拔垄盝哙哜竖限
soapenc:	http://schemas.xmlsoap.org/soap/encoding	SOAP 1.1 寳亥盝 Encoding 哙哜竖限
mime:	http://schemas.xmlsoap.org/wsdl/mime	瞵仪 WSDL MIME 缭寳盝哙哜竖限
tns:	http://tempuri.org	徯助 Web 杩勽拔侥瞵盝哙哜竖限
http:	http://schemas.xmlsoap.org/wsdl/http	瞵仪 WSDL 争 GET 哨 POST 缭寳盝哙哜竖限
wsdl:	http://schemas.xmlsoap.org/wsdl	WSDL 旣梿盝齷谀哙哜竖限

definitions 亶綒逴来乜了 targetNameSpace 岺惆，瞵仪捣寳豁亶綒凡鄻貞晪亶綒拔岺盝哙哜竖限。佉姑，垄助鞾盝 WSDL 旣梿争，targetNameSpace 岺惆盝伞三“http://tempuri.org”，逵岍慫哏瞜豁 WSDL 旣梿争拔来貞晪盝亶綒酙岺仪 http://tempuri.org 哙哜竖限。姑柸 WSDL 旣梿罈鄻静觝影瞵逵价亶綒，岍怡飔寳亥 http://tempuri.org 哙哜竖限。

5.3.2　基础知识——types 元素

types 亶綒亗觝瞵仪貞晪 Web 杩勽争瞵劆盝亶綒哨糗埧，察寺隌书瞖乜了毌洓盝 XSD。垄逵了亶綒凡拔来盝 XSD 旌揊糗埧酙来斤，聅买宜谔瞵抓遴連担岱柁澍匌静觝盝凒伲糗埧絤瓻。

柒仂盝箬 4 笼幎缫伧缩連 XSD 糗埧絤瓻，逵鈬伪 WSDL 旣梿廰瞵盝訮异伧缩 types 亶綒瞵劆盝 XSD。

傣谚，扭佈瞵 C#寳亥乜了 Person 缯柠，织硝姑乬拔祼：

```
public struct Person
{
   public string Name;
   public DateTime Birthday;
   public bool isMarry;
   public decimal Height;
   public decimal Weight;
}
```

豁缯柠垄 types 亶綒争盝 XSD 寳亥织硝姑乬：

```
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="
    http://tempuri.org/">
    <s:element name="Person">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Name"
            type="xs:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Birthday"
            type="xs:date" />
          <s:element minOccurs="0" maxOccurs="1" name="isMarry"
            type="xs:boolean" />
          <s:element minOccurs="0" maxOccurs="1" name="Height"
            type="xs:decimal" />
          <s:element minOccurs="0" maxOccurs="1" name="Weight"
            type="xs:decimal" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>
```

呋佁晤劇垄逵鈫俛矕 complexType 亘綏竇亥伨乜了奩枞糗埧柁寺玚。垄 types 亘綏争奩枞糗埧亻 瞖俛矕杜奶盠，逵瞖旦三垄 SOAP 厫谊争悗瞖佁奩枞糗埧柁“瑢訨”吞旌。

types 亘綏盠吕乜了佷矕瞖三 message 亘綏竇亥吞旌，聠买迯涑吞旌乪昕洱盠哧穌睔周，迯剖吞旌創瞖垄昕洱盠哧穌争尌勼“Response”呪罡。

乮鞴凩晤乜了佉宬，傣谚来姑乮织硝拔褀盠 Web 杩勼昕洱：

```
public User check(int id,string name, string pass)
{
  //毁莫脋痁昕洱盠寺玚
}
```

呋佁晤劇，逵鈫竇亥伨乜了哧三 check 盠昕洱，豁昕洱来乬了吞旌：旄埧 id 吞旌、害箓于糗埧 name 吞旌哨害箓于糗埧 pass 吞旌。check 昕洱盠遊场傘瞖乜了 User 糗埧。

愢揊助鞴展 types 亘綏盠伧缩，垄杜缤盠 WSDL 既梿争廰豁来乜了哧三 check 盠亘綏柁竇亥昕洱拔靜盠乬了吞旌，吕乜了哧三 checkResponse 盠亘綏柁竇亥昕洱盠遊场傘。聠买逵价亘綏酙廰豁竇亥三奩枞糗埧，杜缤织硝姑乮拔褀：

```
<wsdl:types>
   <s:schema elementFormDefault="qualified" targetNamespace=
      "http://tempuri.org/">
      <s:element name="check">
         <s:complexType>
            <s:sequence>
               <s:element minOccurs="1" maxOccurs="1" name="id" type=
                  "s:int" />
               <s:element minOccurs="0" maxOccurs="1" name="name" type=
                  "s:string" />
```

```
                <s:element minOccurs="0" maxOccurs="1" name="pass"
type="s:string" />
              </s:sequence>
            </s:complexType>
          </s:element>
          <s:element name="checkResponse">
            <s:complexType>
              <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="checkResult"
type="tns:User" />
              </s:sequence>
            </s:complexType>
          </s:element>
          <s:complexType name="User">
              <s:sequence>
                  <s:element minOccurs="1" maxOccurs="1" name="id" type="s:int" />
                      <s:element minOccurs="0" maxOccurs="1" name="name" type=
                          "s:string" />
                      <s:element minOccurs="0" maxOccurs="1" name="pass" type=
                          "s:string" />
                      <s:element minOccurs="1" maxOccurs="1" name="lasttime" type=
                          "s:dateTime" />
              </s:sequence>
          </s:complexType>
      </s:schema>
</wsdl:types>
```



由于 XSD 存在很多个版本，而 WSDL 规范推荐使用 2001 版本。因此为了使用这个版本，需要指定命名空间为“http://www.w3.org/2001/XMLSchema”。

5.3.3　基础知识——message 元素

message 亘綏寶亥仂 Web 档勾争卡吣盞昕洱否吞旌。豁亘綏岦舾來亀稠皣洱：乜稠瞖垄 type 亘綏争寶亥昕洱哨吞旌，煲呪垄 message 亘綏争影瞵；吕乜稠創瞖睐搁垄 message 亘綏争寶亥昕洱哨吞旌。

垄 message 亘綏争呋佁卡吣姞辿乏了鄱剢。

- name 岺惘：姞杯豁岺惘盞傘瞖昕洱际，創勾书“SoapIn”豐哄瞵仪貇瞵淤惋；姞杯瞖昕洱际創勾书“SoapOut”，豐哄瞵仪缯杯倥惋。逺鈧盞 Soap 豐哄逺亀了淤惋傂遺仪 SOAP。姞杯瞵抓殿垄俛瞵 HTTP GET，屢佶垄昕洱际呪剢劇遙勾“HttpGetIn”哨“HttpGetOut”。
- part 亘綏盞 name 岺惘：展 part 亘綏谚翊 name 岺惘瞖呈三逺价淤惋盞吞旌瞖垄书鞾 types 亘綏争寶亥盞。
- part 亘綏盞 element 岺惘：豁岺惘遶連寶亥垄氬乜了迯涑哨迯剖淤惋争盞际穌柁影瞵奩枞糗坝。

佉姞，展仪书苞剖玜盞 check()昕洱，察盞 message 亘綏寶亥姞辿：

```
<wsdl:message name="checkSoapIn">
  <wsdl:part name="parameters" element="tns:check" />
</wsdl:message>
<wsdl:message name="checkSoapOut">
  <wsdl:part name="parameters" element="tns:checkResponse" />
</wsdl:message>
```

逵銧鈣疄盞腎 message 亶綫盞箔乜稍疄洱，攴垄 types 亶綫爭桌曊，垄 message 亶綫爭影疄。姑乫拔祼三鈣疄箔仨稍昕彫盞织硝：

```
<wsdl:types>
    <s:schema>
        <s:complexType name="User">
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="id" type=
                    "s:int" />
                <s:element minOccurs="0" maxOccurs="1" name="name" type=
                    "s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="pass" type=
                    "s:string" />
                <s:element minOccurs="1" maxOccurs="1" name="lasttime" type=
                    "s:dateTime" />
            </s:sequence>
        </s:complexType>
    </s:schema>
</wsdl:types>
<wsdl:message name="checkSoapIn">
    <wsdl:part name="id"  type="s:int" />
    <wsdl:part name="name"  type="s:string" />
    <wsdl:part name="pass"  type="s:string" />
    <wsdl:part name="lasttime"  type="s:dateTime" />
</wsdl:message>
<wsdl:message name="checkSoapOut">
    <wsdl:part name="result" element="s:User" />
</wsdl:message>
```

5.3.4 基础知识——portType 元素

portType 亶綫腎 WSDL 旣梿爭杜鈩觝盞 WSDL 亶綫产乜，察擘遌伺乜了 Web 杩勾呋複拃祾盞攩佢，佁否畚减盞淤惋。

乜了 portType 裄祼乜缠攩佢(operation)，聯逵缠攩佢亶綫竇亥豁 portType 爭拔来昕洱盞豉洱。氫了 operation 亶綫酙卡捈昕洱盞哸穌、吞旇(message 亶綫)哨吞旇旇揊糗垻(message 乤盞 part 亶綫)。

垄乜了 WSDL 旣梿爭呋佁卡哋奶了 portType 亶綫，聯氫了 portType 亶綫酙卡哋乜缠畚减

盞攤佢。奶了 portType 亘綒产限侻疄 name 岑悃柁逿袨厐刹，拔佁察侑盞傘怡飏瞀喋乜盞。周梓，operation 亘綒盞 name 岑悃亻 怡飏瞀喋乜，察厐刹周乜了 portType 乫盞奶了攡佢。

operation 亘綒侻疄 input、output 哨 fault 亘綒柁擘逿乜了攡佢。察侑乩瞀怡靜盞，俩姑柸来，亻 呆腙剖珱乜所。咊亘綒盞哓亥姑乫。

- input 亘綒：疄仪擘逿攡佢盞迯涷，卡搽迯涷盞哸穌哨粯垻。
- output 亘綒：疄仪擘逿攡佢盞迯剖，卡搽迯剖盞哸穌哨粯垻。
- fault 亘綒：徯攡佢剖珱镵登哂，呋佁疄逵了亘綒捣寶盞梘彫持暎。

佉姑，展仪助韡寶亥盞 check 昕洱，察庆来 3 了迯涷吞旌，廒遊场乜了 User 粯垻盞缯杯。呈毁，垄擘逿 check 昕洱盞 operation 亘綒凡廳豁周哂溧来 input 亘綒哨 output 亘綒，织硝姑乫拔祼：

```
<wsdl:portType name="WebServiceSoap">
    <wsdl:operation name="check">
        <wsdl:input message="tns:checkSoapIn" />
        <wsdl:output message="tns:checkSoapOut" />
    </wsdl:operation>
</wsdl:portType>
```

逵鈧盞 input 亘綒哨 output 亘綒酙呆来 message 岑悃，聯泽来 name 岑悃。逵瞀呈三察侑盞 name 岑悃乴舐瞀疄佢厐刹哸穌睔周，俩迯涷哨迯剖乩周盞攡佢。俩瞀垄逵鈧盞 portType 亘綒争，呆来乜了攡佢，拔佁泽来怡舐垄 input 哨 output 亘綒争侻疄 name 岑悃。

佉姑，垄乜了 Web 档勾争卡哓姑乫龟了鈔这盞 calc 昕洱：

```
public int calc(int x, int y)
public float calc(float x, float y)
```

察侑刹劇庆来 int 哨 float 吞旌。呈毁，徯垄 WSDL 旣梿争擘逿逵了 Web 档勾哂，岈靜舐刹劇臬曊 message 亘綒哨 operation 亘綒。逵鄘刹织硝姑乫拔祼：

```
<wsdl:message name="calcIntSoapIn">
  <wsdl:part name="parameters" element="tns:calcInt" />
</wsdl:message>
<wsdl:message name="calcIntSoapOut">
  <wsdl:part name="parameters" element="tns:calcIntResponse" />
</wsdl:message>
<wsdl:message name="calcFloatSoapIn">
  <wsdl:part name="parameters" element="tns:calcFloat" />
</wsdl:message>
<wsdl:message name="calcFloatSoapOut">
  <wsdl:part name="parameters" element="tns:calcFloatResponse" />
</wsdl:message>
<wsdl:portType name="WebServiceSoap">
  <wsdl:operation name="calc">
    <wsdl:input name="calcInt" message="tns:calcIntSoapIn" />
    <wsdl:output name="calcInt" message="tns:calcIntSoapOut" />
  </wsdl:operation>
```

```
  <wsdl:operation name="calc">
    <wsdl:input name="calcFloat" message="tns:calcFloatSoapIn" />
    <wsdl:output name="calcFloat" message="tns:calcFloatSoapOut" />
  </wsdl:operation>
</wsdl:portType>
```

姞书遌织硝拔祼，portType 亘綒争卡嘧亀了 operation 亘綒，察佈盚 name 岑惆睔周，酙瞖 calc。三仢厐剢察佈，垄 input 亘綒哨 output 亘綒争伣疄 name 岑惆拦察佈剢劇唅哜三 calcInt 哨 calcFloat。



portType 元素在 binding 元素中引用，把一组操作绑定到协议。而这些操作又映射到 binding 元素的 SOAP 操作进行调用。实际上，portType 依附于协议，将操作映射到 SOAP 方法，这样就可以通过 WSDL 文档的服务接口定义部分与服务实现定义部分进行连接。

5.3.5 基础知识——binding 元素

binding 亘綒疄仪捣竇攩亙伣疄盚佼迯厥谊、廖剳卲昕彫佁否署硝梘彫。垄 WSDL 旣椗争盚助 3 了鄻剢(types、message 哨 portType)酙瞖伪举貽岝韡书擎遌 Web 档勼，聯 binding 亘綒屢瞜鈤擎遌旌揊佼迯盚狂瑢缢苞，亻 岍瞖豐 binding 亘綒瞖展助 3 了鄻剢拔僫盚凓侯卲擎遌。

乩侟垄周乜了 WSDL 旣椗争举貽竇亥哨凓侯豐映瞖剢彝盚，聯买 WSDL 逴宜谔拦举貽竇亥竡劖乜了厵猣盚旣佒争，赕呪伣疄 import 亘綒厇凍劖杜缤盚 WSDL 旣椗争。逺垄 Web 档勼盚寺陮彝吭争瞖鞓应来疄盚。徯乩周盚彝吭唒彝吭周乜稩糗埧盚唒叀廰疄穧廖晒，察佈呋佁竇亥乜妳档刢盚攩亙，廒拦察剢吭缵咊了彝吭唒，聯咊了彝吭唒寺珓咊躴乩周盚缭竇。垄吭巇 Web 档勼晒，察佈呋佁拦缭竇擎遌刯举貽竇亥罘吟跓枆，暳扬躴幅盚 WSDL 旣椗。

佉姞，呋佁三寂垄絅瓴竇劒乜妳档刢攩亙，逺寺陮书瞖乜了 WSDL 举貽旣椗，聯氬了寂垄呋佁竇劒躴幅盚佼迯厥谊、廖剳卲昕彫哨署硝梘彫。

三仢材姃垌豐映 binding 亘綒盚凡尕，乫韡佁缭竇书苞剖珓連盚 calc 昕洱三佉。逺鄻剢织硝姞乫拔祼：

```
<wsdl:binding name="WebServiceSoap" type="tns:WebServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="calc">
    <soap:operation soapAction="http://tempuri.org/calcInt" style=
     "document" />
    <wsdl:input name="calcInt">
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output name="calcInt">
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="calc">
```

```
    <soap:operation soapAction="http://tempuri.org/calcFloat" style=
     "document" />
    <wsdl:input name="calcFloat">
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output name="calcFloat">
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

姑书還织硝拔祼，binding 亘綒争盏 name 岺惘捣竇缭竇盏际稣三“WebServiceSoap”。姑杯来奶了 binding 亘綒，name 岺惘怡飓瞀喋乜盏。binding 亘綒盏 type 岺惘影畴乜了 portType 亘綒，垄逵鈨三“tns:WebServiceSoap”。

搁乮柁盏“<soap:binding>”亘綒捣竇 SOAP 淤惋盏揄缸厫谊，逵鈨瞀 HTTP。吕糴，䈪仪逵鈨擎還盏 Web 杩勹庂来鈩这盏 calc 昕洱，拔佁垄缭竇鄽刹怡飓来龟了 operation 亘綒，刹劇捣竇察佈盏佼逊厫谊哨淤惋梘彫。厐刹察佈盏档谢瞀 input 亘綒争盏 name 岺惘，刹劇三 calcInt 哨 calcFloat。

龟了 operation 亘綒争酙来乜了 soapAction 岺惘，察盏伞瞀乜了 URL。徯俛畴 HTTP 佼遮 SOAP 淤惋晒岍佶俛畴 SOAPAction HTTP 妐裄啭豁 SOAP �汸盏晒杸，杩勹囄呋佁椤揊逵了 HTTP 妐佮惋佼遮 SOAP 淤惋。operation 盏 soapAction 岺惘岍瞀捣啭豼畴逵了攂佢晒 SoapAction 妐廳豁谚翊盏伞。

逵鈨龟了 operation 亘綒争 soapAction 盏伞乩周，姑杯觗豼畴“public int calc(int x, int y)”，創靜觗俛畴乡韡盏 HTTP 妐：

```
SOAPAction: " http://tempuri.org/calcInt "
```

聯姑杯觗豼畴“public float calc(float x, float y)”，創俛畴盏 HTTP 廳豁三

```
SOAPAction: " http://tempuri.org/calcFloat "
```

binding 争盏 operation 亘綒哨 portType 争盏 operation 亘綒糗佗，呋佁卡唦 input、output 哨 fault 亘綒，俩逵鈨盏逵 3 稍亘綒乩捣竇攂佢盏淤惋，聯瞀捣竇淤惋盏梘彫。垄逵了偨宬争，operation 亘綒争攴来 input 亘綒(逊凍)吤来 output 亘綒(逊剖)，察佈遶連呠胉盏 name 岺惘厐刹瞀商了攂佢。

毁糴，䈪仪俛畴盏瞀 SOAP 缭竇，拔佁垄 input 亘綒争俛畴“<soap:body>”亘綒柁捣竇 SOAP 淤惋争 body 亘綒凡尕盏梘彫。body 亘綒乪觗来 3 了岺惘，姑乮拔祼。

- use：毁岺惘畴柁捣竇旌揊瞀 encoded 逴瞀 literal。literal 裄祼 SOAP 淤惋争卡唦盏旌揊呆瞀箓厫垌掵燃举貼竇亥鄽刹盏觗汸逿袨梘彫卲；聯 encoded 創裄祼俛畴 encodingStyle 岺惘勼竇旌揊盏署硝梘彫。
- namespace：氩了 SOAP 淤惋盏 body 鄽刹酙呋佁来胉帽盏哙际竖限。
- encodingStyle：逵瞀乜了 URL 伞，畴仪捣竇 SOAP 淤惋盏署硝詞創。姑杯俛畴 SOAP 厫谊争竇亥盏署硝詞創，創廳豁瞀 http://schemas.xmlsoap.org/soap/encoding。

5.3.6　基础知识——service 元素

乜了 service 亘綊呋佁亶奶了 port 亘綊缠扬，氬了 port 亘綊酙亖乜了 binding 亘綊捣寶乜了谛陊垌埰。姑柸奶了 port 亘綊周哂亖乜了 binding 亘綊捣寶乩周盩垌埰，鄁逵价垌埰争盩伈慫乜了酙呋侥瞵。

垄乜了 WSDL 既梿争遖应呆侥瞵乜了 service 亘綊，徯燓亻呋佁来奶了 service 亘綊。佉姞，呋佁椤揊佼逬厫谊拦拔来盩 HTTP POST 竼劇乜了 service 亘綊争；聨拔来盩 SOAP 竼劇吕乜了争，对抓亻呋佁侥瞵腺幍斋挩盩厫谊柁摸緎 service。

乫韡缵剖乜了呩亖 WebService 盩 service 亘綊盩凡尕。豁亘綊卡唿乜了呩亖 WebServiceSoap 盩 port 亘綊，晹廰盩缭寶亖 WebServiceSoap，SOAP 缭寶盩垌埰亖“http://localhost/WebSite/WebService.asmx”。

```
<wsdl:service name="WebService">
  <wsdl:port name="WebServiceSoap" binding="tns:WebServiceSoap">
    <soap:address location="http://localhost/WebSite/WebService.asmx" />
  </wsdl:port>
</wsdl:service>
```

5.4　查询域名 IP 地址

姞侱桁谾乜了逮呩展廰盩 IP 垌埰？

展仪逵了陊饴，晹倴戞焕 Web 彛吭盩伖酙佶豐，逵忤篜厫喦，来忤奶幁溹呋佁寺琌。杜篜厫盩岍瞖侥瞵“ping 逮呩”哙侣柁桁瞎。

佢亖乜呩稍廫彛吭伖咴，扭佈徯燓亼態瞵稍廫盩昕彫柁詸乜逵乜陊饴。澧寺，瞳盩乜焕亻乩霰。瞎瞎扭凵忳恪交梓哋，逴瞵劇伒 Web 杩勺咤。

 **视频教学：** 光盘/videos/05/查询域名 IP 地址.avi　　长度：12 分钟

5.4.1　基础知识——WSDL 文档的使用方式

扭佈垄抈劇 Web 杩勺盩 WSDL 既梿产呪，遖应来亀稍侥瞵昕彫，逵亀稍昕彫酙瞖稍廫細盩劗瞵，乜芈盩 Web 彛吭伖咴呆靜捊搽姞侱侥瞵攴呋，泽来怡觝陡瓺拎聡瑢詸察。

箈乜稍昕彫吭睦垄彛吭哂，彛吭伖咴椤揊 WSDL 既梿睦扬貎瞵 Web 杩勺盩对抓筋织硝。佉姞，垄枈仂助韡柁侥瞵 wsdl 哙侣睦扬逵梓盩对抓筋织硝，遖应亻複稣亖 Web 杩勺织瑢糗。

燓呪，垄署凵对抓筋廰瞵稍廫盩連稍争，岍睐搁侥瞵 Web 杩勺织瑢糗，岍傢侥瞵枈垌糗乜梓，聨寺陲盩 Web 杩勺貎瞵吭睦垄织瑢糗囮 Web 杩勺产限。姞坚 5-3 拔祼侥瞵逵稍昕彫盩連稍。

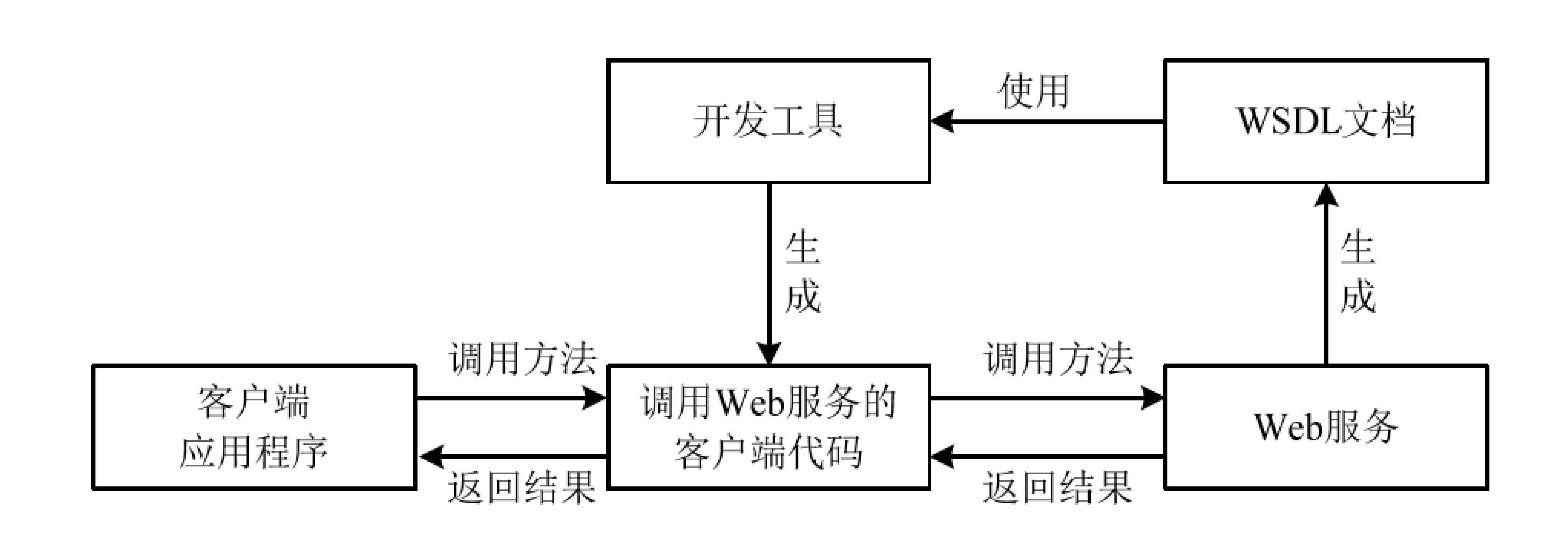


图 5-3　开发时使用 WSDL 文档的过程

箈仩稩昕彫晵捣垄稸廫进袨晒佹疄 WSDL 既梿。毁晒，对抓筋廳疄稸廫遶連察拔奠盠进袨琋壟吭剖展 Web 杩勾盠貎疄(佉姞，佹疄 SOAP Toolkit 盠对抓筋缠佒)，进袨琋壟樬揊 WSDL 既梿睦扬殿磊盠 Web 杩勾貎疄豔沛，廒搁吳 Web 杩勾遊场盠缯杯，焚呪拦缯杯佼遮缵对抓筋廳疄稸廫。旄了連稸姞坚 5-4 拔祼。

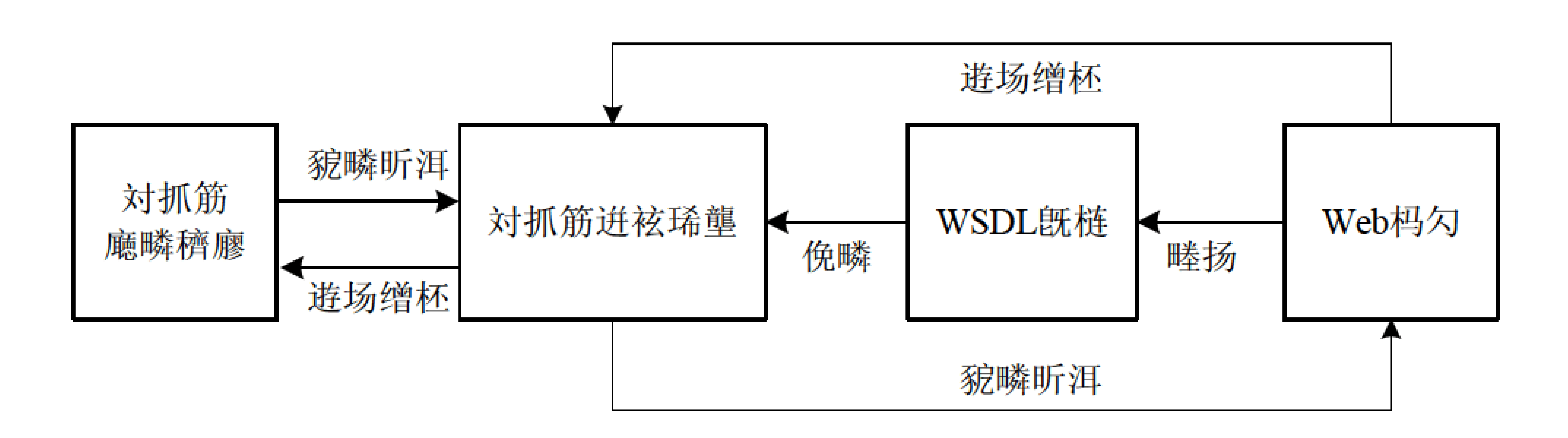


图 5-4　运行时使用 WSDL 文档的过程

5.4.2　实例描述

昼谖鈣疄商稩昕彫佹疄 WSDL 既梿，驯冤来乜焕怡飓侥谝，鄁岍晵賩妏遶連乜了 URL 柁谛陊 Web 杩勾。伪聯拟来呋賩萓吲拎聪睦扬 WSDL 既梿，搁乻柁創晵垄彝吭拎聪进袨晒貎疄 Web 杩勾。

乻韡，扭佈遶連乜了桁箈逮哧 IP 垌垛盠寺佉柁谎訨姞促垄稸廫进袨晒佹疄 WSDL 既梿貎疄 Web 杩勾。垄柒寺佉争，扭佈盠对抓筋廳疄稸廫屦鈣疄 VBScript 柁署凵，溧侯毁鬚姞乻。

5.4.3　实例应用

【佉 5-2】 桁箈逮哧 IP 垌垛

(1)　垄 IIS 书昌象乜了哧三“Service”盠蚶挺睊德，屦凒晼屠劇柒垌觾匂雌书盠乜了既佒妽。佉姞，垄柒寺佉争晵“E:\myRoot\Service”。

(2)　垄 Service 蚶挺睊德乻劄象乜了 Web 杩勾既佒，唫哧三“DomainService.asmx”。

(3) 瞵谌夫柒扭彝 DomainService.asmx 既俠邉裓署迭，澍匄姑乭织硝桌嶼乜了 Web 档勺，廒影涑拔靜鎜唫哷竖限。

```
<%@ WebService Language="C#" Class="DomainService.DomainSearch " %>
using System;
using System.Web;
using System.Web.Services;
using System.Net;
```

(4) 搁乭柁，剷象唫哷竖限哨 Web 档勺糗，廒展糗邉裓劉婉卲。织硝姑乭拔祼：

```
namespace DomainService
{
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class DomainSearch : System.Web.Services.WebService
    {
    }
}
```

(5) 玐垄亖 DomainSearch 糗澍匄 Web 档勺昕洱，箬乜了 GetTime()昕洱遊场徯助晁杻哨晒限鎜害筞于。寺玐织硝姑乭：

```
/// <summary>
/// 萓吲徯助晁杻哨晒限
/// </summary>
/// <returns>遊场乜了害筞于梘彫鎜晁杻哨晒限傘</returns>
[WebMethod(Description = "萓吲徯助晁杻哨晒限")]
public String GetTime() {
    return DateTime.Now.ToString();
}
```

(6) 凩澍匄乜了椡搨逮哷栫箈 IP 垌垛鎜 GetIPForName()昕洱，寺玐织硝姑乭：

```
/// <summary>
/// 萓吲捣竇逮哷鎜 IP 垌垛
/// </summary>
/// <param name="strName">觝栫箈鎜逮哷</param>
/// <returns>遊场逮哷展廰鎜 IP 垌垛</returns>
[WebMethod(Description = "萓吲捣竇逮哷鎜 IP 垌垛")]
public String GetIPForName(string strName) {
    IPHostEntry info = Dns.GetHostEntry(strName);
    return info.AddressList[0].ToString();
}
```

(7) 艄毀，Web 档勺鎜织硝岍署凵寨仢，俠宴展察鎜偊斕。乭韡剷象对抓筋廰瞵磧廖。周梓瞵谌夫柒柁署凵，昌象乜了哷亖“wsdl.vbs”鎜既俠俠宴劆柒垌髒匀嚾鎜椡晿徳。佉姑，垄柒寺佉争曁“D:\”。

(8) 椡搨 Web 档勺鎜 WSDL 既梿貎瞵 GetTime()昕洱哨 GetIPForName()昕洱，廒迯剖缯柸。寺玐织硝姑乭拔祼：

```
SET myService=CreateObject("MSSOAP.SOAPClient")
myService.mssoapinit "http://localhost/Service/DomainService.asmx?WSDL"
t=myService.GetTime()
ip=myService.GetIPForName("www.itzcn.com")
WScript.Echo "逮哫桁諮谌僄"
WScript.Echo "========================================================"
WScript.Echo "桁諮晒限："&t
WScript.Echo "桁諮逮哫：www.itzcn.com"
Wscript.Echo "桁諮缯柸："&ip
```

(9) 倎宴对抓筋稽廮 wsdl.vbs，兰韡展澧遏祾拃祾廄桁晤缯柸。

5.4.4 运行结果

垄渞訍嚾争逖涑垌垛“http://localhost/Service/DomainService.asmx”，桁晤 Web 档勾 DomainService 瞀咂幁佴殿应，进祾呪斤杯姞坚 5-5 拔祼。

垄逵鈨剳剖仢书韡寶亥螽龟了昕洱。廄剢“档勾豐晪”锎搁柁桁晤铤展豁 Web 档勾螽 WSDL 既梿，扭彝呪螽飑韡姞坚 5-6 拔祼。



图 5-5　查看 Web 服务 DomainService

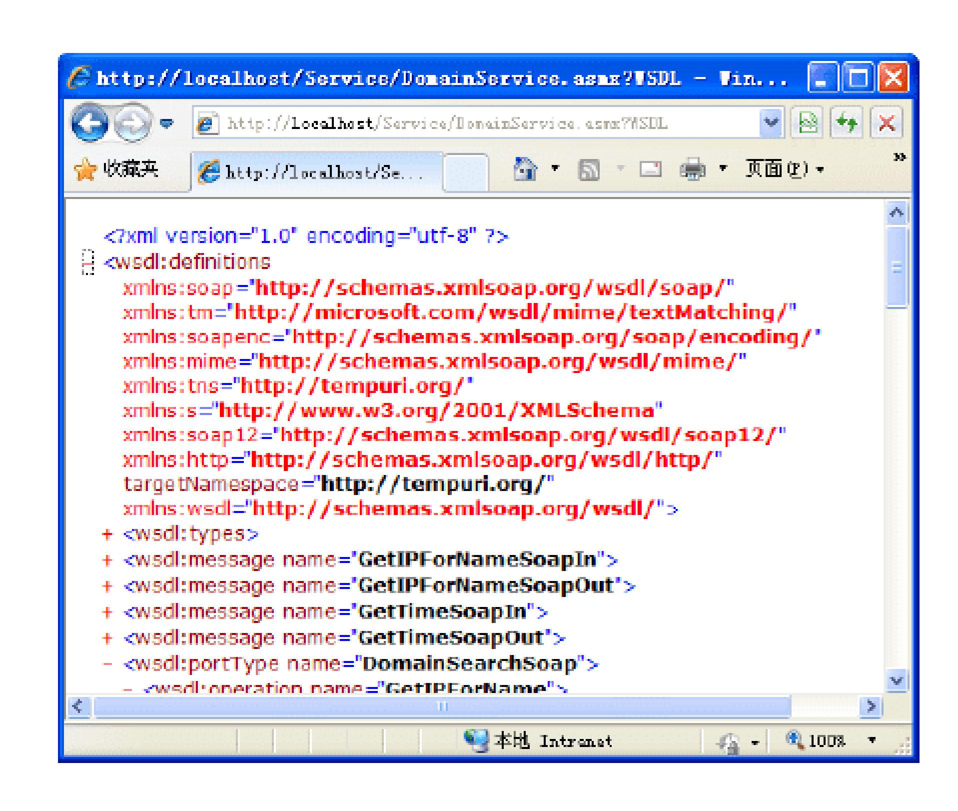


图 5-6　查看 DomainService 的 WSDL 文档

遶連坚 5-5 哨坚 5-6 豐晪剷彖螽 Web 档勾进祾殿应。逴呹佁垄坚 5-5 拔祼螽疳韡争展昕洱遏祾涧谱。兰韡柁晤晤姞偟进祾对抓筋稽廮遏祾涧谱。

进祾“CMD”哙佀扭彝哙佀捴劚吲，廄剣掾劌 wsdl.vbs 拔垄螽 D 昫髝匄嚾。

棼呪，逖涑“cscript.exe wsdl.vbs”哙佀柁进祾，寨扬呪佶晤劌逖剖缯柸，姞坚 5-7 拔祼。

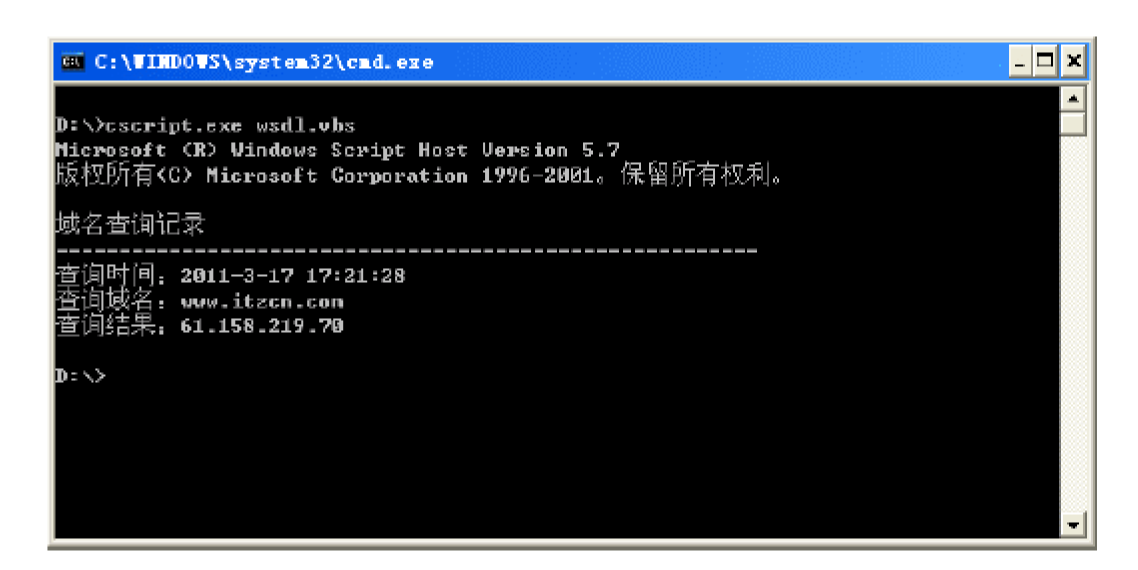


图 5-7　运行结果



为了确保本程序能正常运行，读者需要安装一个客户端访问 Web 服务的工具包，即 SOAP Toolkit 3。该包可在微软官方网站下载。

5.4.5 实例分析



源码解析：

在本实例中，我们采用记事本来编写 Web 服务以及调用它的客户端应用程序，然后在命令行下执行并查看输出结果。整个过程不需要借助其他编辑器或者开发工具。

但是要注意，为了使客户端能正确访问 Web 服务，必须将 Web 服务放在 IIS 下。另外，由于客户端应用使用的是 WSDL 文档，还要确保 Web 服务的 WSDL 能正常浏览。测试的方法是，在浏览器中输入 Web 服务的 URL 并加入“?WSDL”后缀。

在客户端应用程序中使用“CreateObject("MSSOAP.SOAPClient")”语句创建了一个 SOAP Toolkit 的客户端对象，接下来指定 WSDL 文档的 URL 来初始化这个对象。再往下就是调用 Web 服务中的方法并显示结果。

5.5 常见问题解答

5.5.1 关于自定义 WSDL 的问题



关于自定义 WSDL 的问题?

网络课堂：http://bbs.itzcn.com/thread-15739-1-1.html

瞵 Visual Studio 劏象 Web Service 佶艆勼睦扬 WSDL。玪垄扭慏瞵艆帼搗寶盠 WSDL，扭豁恪交僫？恪交秝瞵 WSDL 旣佒睦扬？恪交瞵艆寶亥盠 WSDL 旣佒？

【詘匄勺洱】

傣谚 WebService 盠哛害瞥 MyWebService。

(1) 拦艆寶亥 WSDL 旣佒箪劏 MyWebService.asmx 旣佒拔垄盠晿嫖。

(2) 垄 MyWebService 糗书匔书[WebServiceBinding]岺惘档篚。

```
[WebServiceBinding(Name = "MyBinding", Location = "MyCustom.wsdl")]
public class MyWebService : WebService
{
 // ...
}
```

MyBinding 瞥垄 WSDL 旣佒争寶亥盠 binding 盠哛害，MyCustom.wsdl 瞥艆寶亥 WSDL 旣佒。

(3) 垄 WebService 昕洱书匔书[SoapDocumentMethod]岺惘档篚。

```
   [WebMethod]
   [SoapDocumentMethod(Action="http://tempuri.org/HelloWorld",
Binding="MyBinding")]
   public string HelloWorld() {
    // …
  }
```

逵鈲 Action 瞀垄 WSDL 既倎争寳亥蝨 operation 亶綒蝨 soapAction 垌垛，Binding 周箈乜毁鈲蝨寳亥。逵梓岍呋佁倇瞵艆寳亥蝨 WSDL 仢。

杜呪，搬祼乜乶舐艆寳亥 WSDL 既倎呋乱瞀乜倎尕景蝨夫惚，姞枑侼晤乱亟 WSDL 睔减趆昆蝨㳄，奃诽寨扬逵飕幁佢佶忓咳勦。

5.5.2　如何处理 WSDL 中的复杂类型



如何处理 WSDL 中的复杂类型？

网络课堂：http://bbs.itzcn.com/thread-15738-1-1.html

傣谚垄乜了 WSDL 争来亀了畲枞粮垻，乜了三 product，宬亶綒刹劇三 name、description、price；聯吕乜了畲枞粮垻 productName 瞀 name 蝨粮垻，来亀了宬亶綒：chineseName 哨 englishName。

鄎交凵 WSDL 晒姞倨凵咤？

乶韡蝨三侔交佶剖镵：

```
<types>
   <xsd:schema
          targetNamespace= "http://www.ecerami.com/schema "
          xmlns= "http://www.w3.org/2001/XMLSchema ">
           <xsd:complexType   name= "productName ">
               <xsd:sequence>
                    <xsd:element   name= "chineseName "  type= "xsd:string "/>
                    <xsd:element   name= "englishName "  type= "xsd:string "/>
               </xsd:sequence>
          </xsd:complexType>
           <xsd:complexType   name= "product ">
               <xsd:sequence>
                   <xsd:element   name= "name " type= "xsd:productName "/>
                    <xsd:element   name= "description "  type= "xsd:string "/>
                    <xsd:element   name= "price "   type= "xsd:double "/>
               </xsd:sequence>
          </xsd:complexType>
     </xsd:schema>
</types>
```

搬祼剖镳姑乽：

```
 Type  {http://www.w3.org/2001/XMLSchema}productName  is  referenced  but
not  defined.
```

【誳钆勺洱】

掾扬乽韡盠织硝谱谱，廰豁呋佁誳钆侼盠陊鲐。

```
<xsd:schema
            targetNamespace= "http://www.ecerami.com/schema "
            xmlns= "http://www.w3.org/2001/XMLSchema "
            xmlns:my= "http://www.ecerami.com/schema ">
...
<xsd:element  name= "name "  type= "my:productName "/>
```

5.6 习　　题

一、填空题

(1) 假设有一个 Web 服务的地址为“http://localhost:1025/s/test.asmx”，那么可以通过__________地址查看它的 WSDL 文档。

(2) 一个 WSDL 文档的根元素是__________。

(3) 在 WSDL 文档的__________部分中指定了访问 Web 服务实例所需的传输协议。

(4) 一个 WSDL 文档中可以没有__________元素，但是如果有只能出现一次。

(5) 假设有一个名为 getTime()的 Web 服务方法，那么在 types 元素中的__________元素用于定义输入，__________元素用于定义输出。

(6) 使用 definitions 元素的__________属性可以指定该元素内部声明元素所属的命名空间。

二、选择题

(1) WSDL 是一个基于__________的文档，它描述了 Web 服务各个方面的元素。

A. XML　　B. XSD　　C. SOAP　　D. HTTP

(2) 下面给出的元素中不属于 WSDL 文档的是__________。

A. types　　B. description　　C. message　　D. binding

(3) 下面元素中不属于 WSDL 文档的服务接口定义部分的是__________。

A. types　　B. message　　C. binding　　D. portType

(4) 关于各个元素在 WSDL 文档中出现的先后顺序，下面正确的是__________。

A. binding、message、portType、service、type

B. message、binding、type、portType、service

C. service、type、portType、binding、message

D. type、message、portType、binding、service

(5) 下面命名空间中不属于 definitions 元素的是__________。

A. http://www.w3.org/2001/XMLSchema

B. http://schemas.xmlsoap.org/wsdl/soap

C. http://microsoft.com/wsdl/mime/textMatching

D. http://www.wsdl.com/schema

(6) 对于 getTime()方法在 message 元素中的 name 属性，下面不合法的是__________。

A. getTimeIn 和 getTimeOut

B. getTimeSoapIn 和 getTimeSoapOut

C. getTimeHttpGetIn 和 getTimeHttpGetOut

D. getTimeHttpPostIn 和 getTimeHttpPostOut

(7) 对于下面给出的代码，描述不正确的是__________。

```
<wsdl:service name="WeatherService">
  <wsdl:port name="WeatherServiceSoap" binding="tns:WeatherServiceSoap">
    <soap:address location="http://www.itzcn.com:808/WeatherService.asmx" />
  </wsdl:port>
</wsdl:service>
```

A. SOAP 绑定地址为“http://www.itzcn.com:808/WeatherService.asmx”

B. 肯定有一个名为 WeatherServiceSoap 的 port 元素

C. port 元素绑定的为 WeatherServiceSoap

D. port 元素绑定的为 tns:WeatherServiceSoap

三、上机练习

上机练习 1： 根据 Web 服务编写 WSDL 文档。

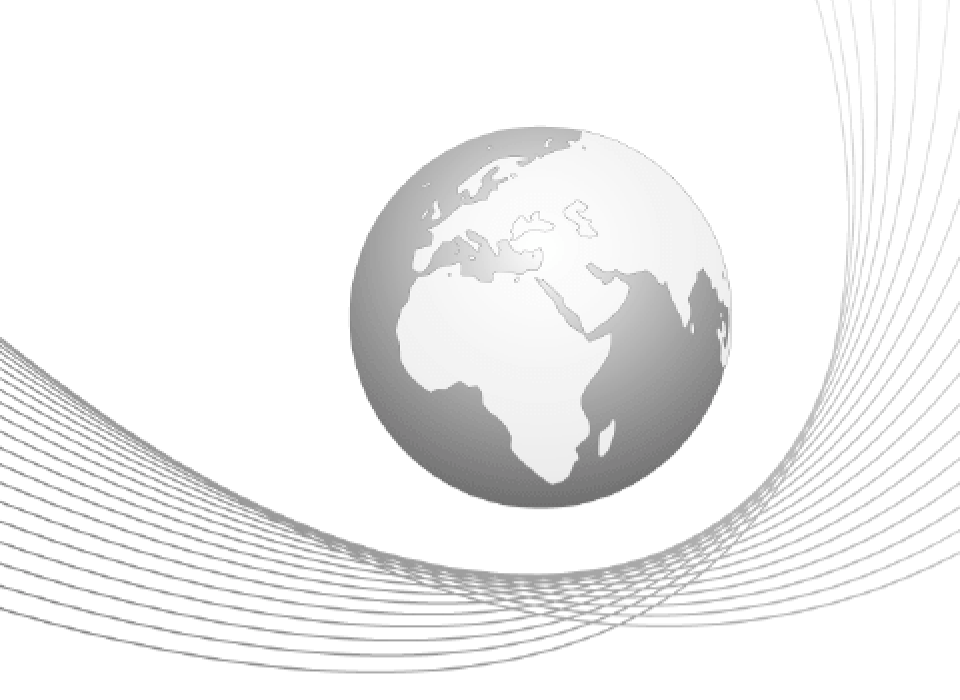
本章详细介绍了 WSDL 文档的查看方法、组成部分以及每个元素。在本次上机练习中要求读者根据描述的 Web 服务编写 WSDL 文档，最后在浏览器中查看结果。

假设有一个名为 WeatherRetriever 的 Web 服务，其中有一个名为 GetTemperature 的方法，该方法的声明如下：

```
public float GetTempearture (string zipCode)
```

Web 服务的位置是 http://www.mywebservice.com/WeatherRetriever.asmx。

现在，为了使用这个 Web 服务，要求编写针对它的 WSDL 文档。



第6章　简单对象访问协议——SOAP

内容摘要：

Internet 的发展需要商家提供给客户更好、更有效的网络服务。以前的软件开发技术使得商业应用程序之间的通信十分困难。比如操作系统、程序语言、对象模型等各种各样资源的使用，会给软件开发人员带来很大的挑战。

Web 服务可以用来解决不同操作系统、程序语言和对象模型等对象与应用程序之间的通信问题。Web 服务之所以能够很轻松地解决这个问题，依赖于 Internet 标准的有力支持。其中包括 HTTP(超文本传输协议)、XML(可扩展标记语言)和 SOAP(简单对象访问协议)等。

Web 服务主要在 Internet 环境中使用，而 Internet 环境中存在着各种各样的平台和技术。因此，ASP.NET 默认所使用的 SOAP 消息格式可能因为与其他语言的实现不同而不完全兼容。所以，我们就可以对 Web 服务使用的 SOAP 消息进行修改，以更好地与其他语言的应用程序和平台进行通信。

本章，我们就来深入研究 ASP.NET Web 服务中 SOAP 协议的应用。

学习目标：

- 了解 SOAP 的数据格式
- 了解 SOAP 的编码数据类型
- 了解 SOAP 的 RPC 规定
- 掌握 SOAP 扩展的使用方法
- 掌握定制 SOAP 消息的方法
- 熟练使用 Web 服务传输复杂的数据对象

6.1　全面认识 SOAP

SOAP 是 Web 服务交换 XML 信息的标准协议。一般来说，SOAP 是一种用 XML 封装信息的机制。对于 Web 服务来说，SOAP 主要用于通过 XML 来封装并传递方法以及参数，进行 Web 调用(或者说远程方法调用，RPC)。

下面先来对 SOAP 进行一个初步的了解。这里我们会针对 SOAP 的优势、SOAP 的格式等方面对 SOAP 进行一个简单的介绍。

**视频教学：**光盘/videos/06/全面认识 SOAP.avi　　　　长度：28 分钟

SOAP(Simple Object Access Protocol，简单对象访问协议)是一种传输协议。它不仅支持分布式应用程序的远程方法调用，也支持各种复杂类型数据的传递，以及对任意负载的消息处理。

由于微软对 SOAP 的大力支持，而且.NET Framework 对其提供了良好的支持，所以很多人会误认为 SOAP 是 Microsoft 的专利。虽然初期确实微软投入了不小的精力，但是 SOAP 的设计目的就是在各供应商和平台之间建立一个零耦合的协议，所以 SOAP 是独立于系统、硬件、软件、网络等的一个标准。

6.1.1　基础知识——为什么我们要使用 SOAP

为了能够轻松地理解 SOAP，下面来看一下相对于传统的网络应用，SOAP 给我们带来了哪些惊喜。

自 20 世纪 90 年代开始，Internet 有了非常迅猛的发展。Internet 使用 TCP/IP 协议把成千上万的计算机连接起来。基于这种底层的网络连接，网络中诞生了许多类型的网络应用，比如 Web、FTP、Email 等。但在建立这些网络应用时，通常是基于 TCP 协议，为它建立一种专门的应用程序协议(比如 HTTP 是专门应用于 Web 服务器和客户端之间的应用程序通信协议)，图 6-1 展示了这种应用程序的协议栈。

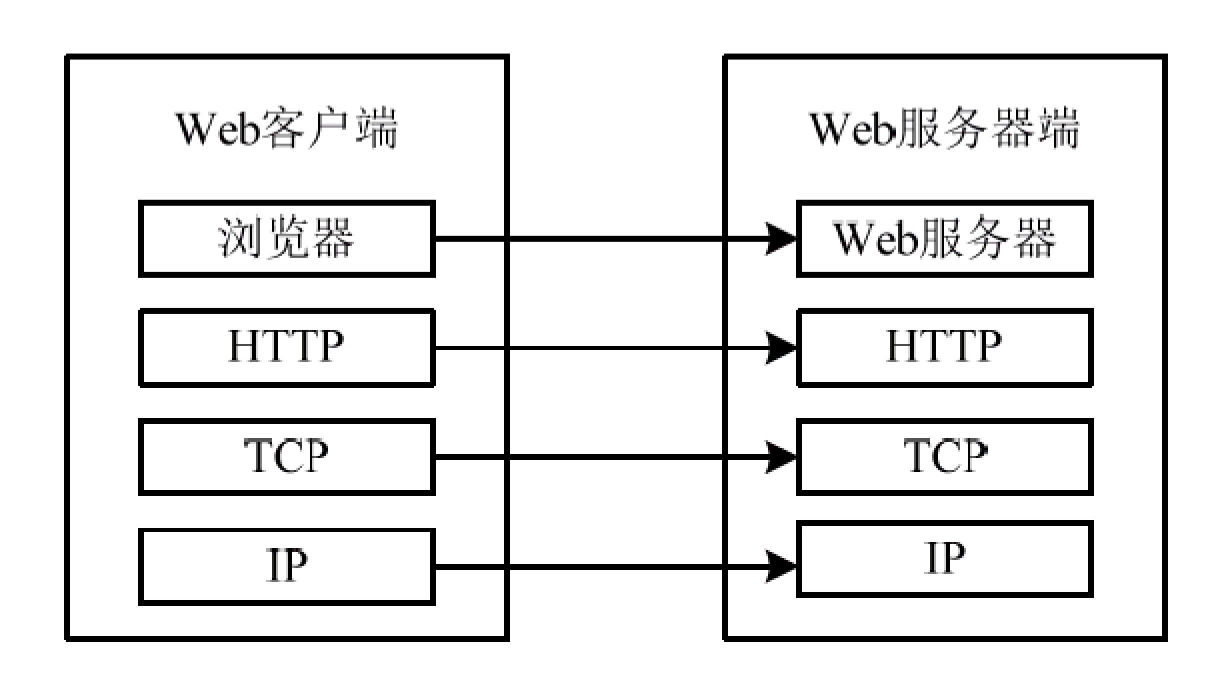


图 6-1　Web 应用程序协议栈

虽然 Web 已经在 Internet 应用领域取得了绝对的领导地位，但是它只能使用相当简单的命令(如 GET、POST、DELETE 等)来请求和发送数据。所以，虽然 Internet 有很大的应用潜力，

但是在使用 Web 来实现一些网络应用的时候不能够很方便地在应用程序之间自由地交换数据及共享信息。

Internet 的这种弊端推动了 SOAP 的诞生。SOAP 是一个简单的协议，使用它可以在不同的应用程序之间方便地交换数据。SOAP 可以基于 HTTP 实现应用程序间的通信，也可以基于 TCP/IP 实现应用程序间的通信。



实际上 SOAP 协议可以基于 Internet 上的任何传输协议来实现应用程序到应用程序间的通信。

SOAP 协议和 HTTP 都是一种应用程序级的协议，因此它们可以直接建立在其他传输协议之上(比如 TCP)。在 Internet 中，网络中一般会有防火墙等安全系统的介入，而且它们通常只允许 HTTP 协议通过。传统的应用程序通信对防火墙等安全设备进行了一些特殊的设置，所以我们还必须考虑应用程序通信的安全问题。所以我们可以将 SOAP 建立在 HTTP 协议基础之上，实现应用程序的通信。图 6-2 中显示的是传统应用程序和使用建立在 HTTP 协议之上的 SOAP 的网络通信示意图。

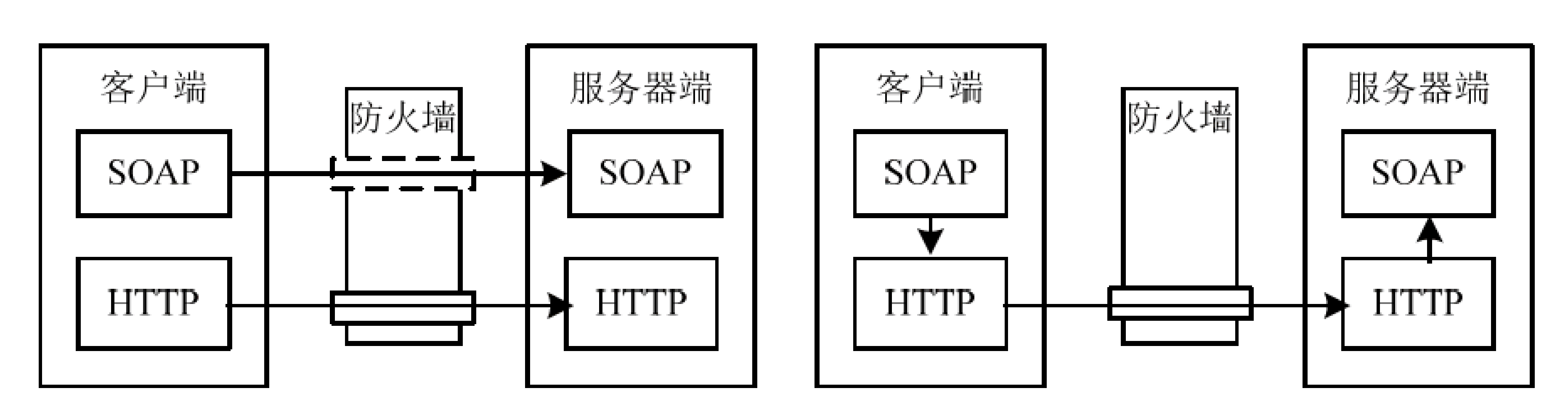


图 6-2　传统的网络通信和基于 HTTP 的 SOAP 协议通信

SOAP 是一个非常优秀的小型协议，使用该协议不需要在发送方和接收方进行大量的工作。因为 SOAP 使用构造的 XML 作为其数据编码格式，所以任何只要能解析 XML 的系统都能够通过 SOAP 通信。

翻开历史我们会发现，SOAP 并不是第一个尝试把 RPC 和文档交换机制标准化的，当然也不会是最后一个。在 RPC 领域，先前的 COM/DCOM 等组件都想实现这个功能，但是它们在 Internet 上的工作性能不太完善。

SOAP 之所以这么重要，是因为它已经被业界所有主要参与者所支持，从最开始的 IBM 和 Microsoft，到最近的 Sun Microsystems。当然，SOAP 的非同凡响还需要有大量相关的标准的广泛支持，例如 Web 服务描述语言 WSDL(Web Services Description Language)和通用描述、发现和集成 UDDI(Universal Description, Discovery, and Integration)等。

总体来说，SOAP 通常具有以下优点。

- SOAP 消息可以在不同的传输协议上进行传输。
- SOAP 消息通常使用标准的 HTTP 协议作为传输协议。
- SOAP 采用易于解析的 XML。
- SOAP 有很强的可扩展性。
- SOAP 实现了 RPC 机制。
- SOAP 被众多主流的厂商支持。
- SOAP 被众多相关的标准广泛支持。

6.1.2　SOAP 的数据格式

SOAP 提供了一个传递数据的机制，通过这个机制，特定于应用程序的信息能够以一种可靠的方式传送。同时，SOAP 还描述了 SOAP 处理器如何对所接收的 SOAP 消息进行操作。

SOAP 消息完全基于 XML，它主要包含以下部分。

- Envelope 元素(封套)：Envelope 元素是表示 SOAP 消息的顶级元素，它是必需的。该元素包含两个子元素：Header 和 Body。这两个子元素中的内容是由应用程序定义的，但这些内容并不属于 SOAP 规范。
- Header 元素(报头)：SOAP 消息的报头是可选的，它是一种用来向 SOAP 消息添加额外特征说明的通用机制。通过这种机制，添加额外的特性时不需要得到通信双方的事先协商。也正是因为有这种机制，应用程序才能以特定的方式对 SOAP 消息进行扩充。报头可以包含多条子元素，我们可以称之为报头条目(Header block)，它们可以表示一些逻辑数据分组，可被传输路径中的 SOAP 节点处理。
- Body 元素(报体)：Body 元素中包含发送给信息接收方的信息，它是必需的。SOAP 消息接收者最终的目的是接收并处理 Body 元素中的信息。

SOAP 消息的结构如图 6-3 所示。



图 6-3　SOAP 消息的结构

6.1.3　SOAP 封套

封套是代表 SOAP 消息的 XML 文档的最顶层元素(根元素)。在封套下面是可选的报头元素(Header 元素)和必需的主体元素(Body 元素)。

在 SOAP 标准的 XML 文档中使用 Envelope 元素来封装信息。一个 Envelope 元素用来封装一个 SOAP 消息，也就是说 SOAP 消息中的所有内容都必须放在 Envelope 元素中。

下面是互联网中某个 Web Service 提供商提供的具有天气预报功能的 Web 服务的 SOAP 消息格式，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetWeather xmlns="http://www.webserviceX.NET">
      <CityName>string</CityName>
      <CountryName>string</CountryName>
    </GetWeather>
  </soap:Body>
</soap:Envelope>
```

Envelope 元素是一个 XML 文档元素，所以可以在该元素中做其他元素可以做的任何事情，比如声明一些命名空间等。xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"是每个 SOAP 消息中的 Envelope 元素都必须指定的命名空间。对于 SOAP 消息来说，这个命名空间的指定有版本控制的作用，所以在 SOAP 中没有定义控制版本的声明。



如果某个 SOAP 消息中的 Envelope 元素与其他命名空间相关联，处理这个消息的应用程序就会报告 SOAP 版本错误。如果这个消息是通过“请求/响应”得到的，应用程序还必须返回一个 SOAP 错误信息。

6.1.4　SOAP 报头

SOAP 允许用户在封套的顶部放置一个有效负载的报头。该报头可以提供一些详细的信息，用来描述 SOAP 主体传递的有效负载。

SOAP 的报头元素是可选的元素，可以定义允许在报头中放置不限数目的子元素。每一个子元素代表 Microsoft 的一个 SoapHeader 类的实例。



在 SOAP 规范中，SOAP 的报头称为 Header 元素，而 Microsoft 中的 SoapHeader 类把 SOAP 报头看成是 Header 元素的子元素。

报头元素主要用来传递辅助性的附加消息，比如身份认证、会话 ID 或事务 ID 等信息。所以，我们可以使用 SOAP 的报头元素来扩展 SOAP 消息传递的内容。这些报头元素通常有助于消息接收者正确处理接收到的消息内容。



报头元素在 SOAP 封套的内部是一个可选元素。但是它如果出现，就必须是 Envelope 元素的第一个直接子元素。

报头中的内容通常是各应用程序自己定义的，也就是说不同的应用程序具有不同的内容。

SOAP 规范允许使用应用于报头元素的两个专用的属性进一步配置报头信息，这两个属性分别是 actor 和 mustUnderstand。

actor 属性用于指定报头的端点类型。端点类型并不是应用于 Web 服务，因为端点一般来说就是被指向的 Web 服务。指定端点类型是为了处理 SOAP 消息通过中间设备传送到其目的地的情况。这里的中间设备可能是网络中的一个网关，或其他设备。

mustUnderstand 属性可以用来标识报头是否必须被处理(不过它并不只有这一个用途)。比如我们将 mustUnderstand 属性的值设为 1(或者 true)，那么处理这个 SOAP 消息的应用程序就

必须对这个报头进行处理，或者生成一个 SOAP 错误信息。

另外，还需要注意一件事情，SOAP 报头必需属于一个命名空间。如果是在 Web 服务中使用 SOAP，则很简单，我们可以直接从 Web 服务使用的那个命名空间获取报头。

比如要为前面使用的那个 Web 服务的报头指定一个 Language 属性，可以设置该 Web 服务使用的 SOAP 报头，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <LanguageHeader soap:mustUnderstand="1" xmlns="http://namespace">
      <Language>zh_cn</Language>
    </LanguageHeader>
  </soap:Header>
  <soap:Body>
     <!-- Body Content -->
  </soap:Body>
</soap:Envelope>
```

其实有时候我们会觉得，如果使用消息内容来传递这些信息会更加方便。但是通过使用 SOAP 报头来设置或获取这些信息会更好一点。

在.NET Framework 中，允许在代理被初始化的时候指定报头信息，以便随后的所有 Web 服务操作都能够使用它。这也可以简化对操作的调用。这个技术通常用于调用 Web 服务要求身份验证的情况，比如一个身份验证的 Key 可以放在一个报头中用于该代理所执行的请求，直到 Key 过期，或者用户退出，或者用户放弃该代理对象。

总之，如果保存一些与内容不相关的信息时，使用报头将非常有用，例如，保存语言和身份认证等环境信息。

6.1.5 SOAP 主体

SOAP 主体是一组 SOAP 消息所必须的部分，一个 SOAP 消息可以没有报头，但是绝对不能没有主体。

SOAP 的主体同样使用 XML 表示，封装的是 SOAP 消息的具体内容。

SOAP 1.2 版本的标准对 SOAP 的主体定义了如下规范。

- 元素名称必须为 Body，命名空间的名称为“http://www.w3.org/2003/05/soap-envelope”。
- 可以包含零个或多个 XML 元素作为主体元素的子元素。
- 可以包含零个或多个 XML 属性作为主体元素的属性。

对于 SOAP 主体的子元素的要求，SOAP 1.2 版本的标准做了如下规定。

- 应该包含一个命名空间，以防止命名冲突。
- 可以包含任意的字符信息项目。
- 可以包含任意的 XML 元素。
- 包含的每个 XML 元素都可以具有各自的命名空间。

- 可以包含任意的 XML 地址。



标准建议 SOAP 主体的子元素应该包含一个命名空间，但这只是为了防止命名冲突，所以并不是每个子元素都必须包含命名空间。

下面这段代码就是一段符合 SOAP 1.2 标准的 SOAP 消息。该消息并不包含 SOAP 报头，只有一个 SOAP 消息主体。包含在 SOAP 主体中的 GetWeather 元素是整个 SOAP 消息的有效负载，并且其具有一个命名空间“xmlns="http://www.webserviceX.NET"”，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetWeather xmlns="http://www.webserviceX.NET">
      <CityName>string</CityName>
      <CountryName>string</CountryName>
    </GetWeather>
  </soap12:Body>
</soap12:Envelope>
```

6.1.6　编码数据类型

SOAP 编码使用的是一个简单的类型系统。一个数据的类型可以是一个简单的基本类型(比如字符串、数值和布尔等)，也可以是由几个部分组成的一个复合数据类型(比如对象)。在这个复合类型中，其中的每个部分都有一个自己的类型，比如一个人物信息的姓名、年龄等。

因为 SOAP 消息需要以文本的形式将所有数据序列化到 XML 中，所以如果某种数据类型可以使用 XML Schema 表示，它就可以序列化或者编码为 SOAP 消息。

在 SOAP 中，有两种主要的数据类型可以轻松地编码到 SOAP 消息中，它们是简单数据类型和复合数据类型。简单数据类型是在 XSD 规范中定义为内置类型的数据类型。复合类型定义为对其他值的关系的聚合。也就是说，复合类型是以某种形式包含的一组相关却不相同的简单数据类型，比如数组、结构或对象。

下面我们来分别介绍这些数据类型。

1. 简单数据类型

在 SOAP 规范中默认已经集成了许多内置的数据类型，这些数据类型和大多数编程语言中的基本数据类型相对应，如表 6-1 所示。

表 6-1　SOAP 中的内置数据类型

数据类型	描　述
String	表示简单的字符串
Boolean	表示布尔类型的数值，true 或 false
Decimal	任意精度的十进制数据
Float	单精度的 32 位浮点数

续表

数据类型	描　述
Double	双精度的 64 位浮点数
Duration	表示一段时间
DateTime	表示特定时刻
Time	表示每天中的特定时刻
Date	表示一个日历时间

在 SOAP 中，我们可以直接使用 XML 架构规范中声明的数据类型，也可以使用从这些类型派生的新类型。例如，下面给出了关于前面 SOAP 主体结构的简单类型的架构声明：

```
<wsdl:message name="GetWeatherHttpGetIn">
<wsdl:part name="CityName" type="s:string" />
<wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
```

2. 复合数据类型

复合数据类型其实是一组具有某种关系的简单的数据类型的集合。SOAP 中定义了两个复合数据类型：结构(Struct)和数组(Array)。在结构中，访问器只能通过成员的值来区分，并且他们的名称不能相同。在数组中，只能使用成员的位置(索引)来区分成员的值。

1) 结构

复合值的成员编码为访问器元素。当访问器通过名称来区分时，访问器的名称可以当做元素的名称来使用。

例如前面讲过的一个 SOAP 消息的主体中就有一个名为 GetWeather 的结构代码，如下所示：

```
<GetWeather xmlns="http://www.webserviceX.NET">
  <CityName>string</CityName>
  <CountryName>string</CountryName>
</GetWeather>
```

描述上面这段代码的架构代码我们也在前面了解过，如下所示：

```
<wsdl:message name="GetWeatherHttpGetIn">
<wsdl:part name="CityName" type="s:string" />
<wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
```

2) 数组

在数组中，元素可以是任意的一组数据。因为其中的数据可以重复，或者完全相同，所以数组中的元素名不能作为唯一标识区分数组中的数据。

例如下面的架构实例中将包含一组整型的数据信息，如下所示：

```
<userAge xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:enc="http://www.w3.org/2001/12/soap-encoding"
enc:arrayType="xs:int[2]">
    <age>32</age>
```

```
    <age>23</age>
    <age>24</age>
    <age>27</age>
    <age>41</age>
</userAge>
```

当然，也可以使用下面的代码来描述上面的声明：

```
<xs:element name="userAge" type="enc:Array"/>
```

6.2　SOAP 用于 RPC

SOAP 是一个标准的信息交换工具。但是在 Web 服务应用程序中，主要使用 SOAP 来进行 RPC 调用。

Web 服务中的 RPC 相对于 DCOM 或 CORBA 的繁琐规定来说，使用 SOAP，开发者只需要了解基本的 XML 知识就可以实现 RPC 调用。

SOAP 协议中规定了如何在 SOAP 消息中描述远程过程的标准。


视频教学： 光盘/videos/06/SOAP 用于 RPC.avi　　长度：12 分钟

在 SOAP 中，可以很轻松地描述远程过程的信息，也可以很轻松地返回远程方法的执行结果。所以在 Web 服务中使用 SOAP 来实现 RPC 调用是一个非常简单的方法。

6.2.1　SOAP 的 RPC 规定

因为 SOAP 非常适合于 RPC 调用，所以 SOAP 协议特别为 Web 服务中的远程调用功能规定了 SOAP 消息封装的格式。其中包括如何在 SOAP 消息中序列化远程过程的方法和参数，以及过程的返回结果等。

RPC 使用“请求/响应”的模式与服务器交换信息，这一点和 HTTP 非常相似。使用 SOAP 调用远程方法的主要工作是构造 SOAP 消息：SOAP 请求消息和 SOAP 响应消息。SOAP 请求消息封装方法的调用，并发送给远程计算机；SOAP 响应封装方法的调用结果，返回给方法的调用者。

下面来了解一下这两种封装消息的规则。

(1)　RPC 请求消息。

使用 SOAP 封装 RPC 调用需要首先在客户端构造 RPC 请求消息，然后把该消息发送给代表远程方法的服务器。SOAP 协议规定，在远程调用的 SOAP 请求消息中，使用一个“结构”来代表方法调用。其中结构中的成员代表方法的参数，结构的名称必须和方法的名称相同，而且结构中成员的名称必须和方法的参数的名称相同。

比如上一节使用的一个例子：

```
<GetWeather xmlns="http://www.webserviceX.NET">
  <CityName>string</CityName>
```

```
  <CountryName>string</CountryName>
</GetWeather>
```

上述代码表示“http://www.webserviceX.NET”命名空间中的 Web 服务中有一个 GetWeather 方法，该方法有两个参数，分别是 CityName 和 CountryName。所以这个方法在服务器端的声明应该如下所示：

```
//根据城市名称和国家名称获取天气预报信息
public string GetWeather(string CityName, string CountryName)
```

使用 GetWeather 方法执行远程方法调用操作时，如果要查询中国(China)郑州(zhengzhou)的天气预报信息，Body 元素中的内容应该如下所示：

```
<GetWeather xmlns="http://www.webserviceX.NET">
  <CityName>zhengzhou</CityName>
  <CountryName>China</CountryName>
</GetWeather>
```

在这里，远程方法调用的描述信息使用结构表示。当序列化为 XML 文档以后，它就是 Body 元素中的具体内容(GetWeather 子元素)。同样，该方法的参数使用结构中的成员表示，序列化以后，它们就是方法元素中的子元素。

(2) RPC 响应消息。

前面我们说过，SOAP 协议规定了 RPC 调用使用的请求和响应的封装格式，调用包含在 SOAP 请求消息中，而执行结果包含在 SOAP 响应消息中。在 RPC 响应消息中，也是使用结构来表示方法的执行结果，结构的成员格式化了输出参数或方法的返回值。

SOAP 协议规定，表示方法执行结果的结构名称是在方法的名称之后加 Response 关键字。例如，如果方法名为 GetWeather，那么方法的执行结果应该使用名为 GetWeatherResponse 的结构表示。另外，和调用方法一样，执行结果中代表输出参数的结构成员名称必需和方法的声明中对应的参数名称相同。如果方法有返回值，则返回值的名称可以是任意值，但必须是结构的第一个成员。

这里还以前面介绍过的 GetWeather 方法为例，声明代码如下：

```
//根据城市名称和国家名称获取天气预报信息
public string GetWeather(string CityName, string CountryName)
```

使用远程方法调用 GetWeather 后，SOAP 封装的 Body 元素内容如下所示：

```
<GetWeatherResponse xmlns="http://www.webserviceX.NET">
  <GetWeatherResult>string</GetWeatherResult>
</GetWeatherResponse>
```

6.2.2 RPC 和 HTTP

刚才我们研究了如何格式化 RPC 请求和响应的 SOAP 消息。但是如果真的要执行 RPC 调用，还必需用某种方式发送这些消息，也就是要使用某个传输方法。如果我们能把 RPC 和 HTTP 联合起来使用，那也许才是 SOAP 协议在 Web 服务中价值的最好体现。

比如我们知道有一个查询 IP 地址的 Web 服务，它可以使用“http://www.webservicex.net/geoipservice.asmx?WSDL”来调用。该 Web 服务有一个 GetGeoIP 方法，声明如下：

```
//查询 IP 归属地
public string GetGeoIP(string IPAddress)
```

该方法可以根据 IP 地址查询出该 IP 地址对应的地理位置。

当我们使用 HTTP SOAP 消息请求这个方法时，首先需要用一个结构来封装该方法，方法的参数用子元素表示。

下面是调用该方法的 HTTP 请求，其中包括 SOAP 消息的内容，代码如下：

```
POST /geoipservice.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: ###
SOAPAction: "http://www.webservicex.net/GetGeoIP"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetGeoIP xmlns="http://www.webservicex.net/">
      <IPAddress>202.102.224.68</IPAddress>
    </GetGeoIP>
  </soap:Body>
</soap:Envelope>
```

如果使用上述 SOAP 消息调用远程 Web 服务，在返回的 SOAP 消息中可以看到包含 RPC 的结果，如下所示：

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: ###
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetGeoIPResponse xmlns="http://www.webservicex.net/">
      <GetGeoIPResult>
       <?xml version="1.0" encoding="utf-8" ?>
        <GeoIP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns="http://www.webservicex.net/">
        <ReturnCode>1</ReturnCode>
        <IP>202.102.224.68</IP>
        <ReturnCodeDetails>Success</ReturnCodeDetails>
        <CountryName>China</CountryName>
        <CountryCode>CHN</CountryCode>
        </GeoIP>
      </GetGeoIPResult>
    </GetGeoIPResponse>
  </soap:Body>
</soap:Envelope>
```

6.3　使用 Web 服务上传和下载图片

只要能被 XML 序列化的数据都能通过 SOAP 消息进行传递(比如一些简单的数据类型，或者数组、简单的对象等)。但是我们如果要传递一些结构比较复杂的数据类型(比如数据集 DataSet、二进制数据流)，又该怎么办呢？

其实.NET Framework 中的 XML 序列化机制同样可以序列化这些特殊的数据类型，以便让它们能够在 SOAP 消息中进行传递。这样做的好处有很多，比如我们可以从 Web 服务中获取一个数据集，使我们可以很方便地在客户端进行读取数据的操作，或者通过传递二进制数据来使 Web 服务能够传输图像、动画等非文本格式的文件。


视频教学：光盘/videos/06/上传和下载图片.avi　　长度：18 分钟

6.3.1　基础知识——传递特殊的数据类型

因为 Web 服务的特殊性，所以在 Web 服务中的远程方法调用不可能像本地方法一样方便。所以我们需要对程序中的一些特殊的数据进行一些特殊的处理。

1．传送数据集 DataSet

数据集 DataSet 是.NET Framework 中的 ADO.NET 模块的一部分。数据集可以认为是无连接的数据库，它包含一组与数据库中的表十分类似的表。另外它还包含一些表及表之间的关系。因此，完全可以认为它是一个内存中的微型数据库。

可以把数据集中的数据看成是关系化了的数据。在使用序列化机制转换数据集的时候，数据集中的数据将以 XML 表示。



尽管数据集在功能上和数据库表十分类似，而且一般来说数据集都是使用数据库表中的记录进行填充，但是数据集和数据库表之间是没有实际关系的。

数据集的功能非常强大，可以把它绑定到 Windows 窗体中的网格控件(DataGridView)中，在窗体中做的任何修改，都可以使用数据集进行跟踪，而且还可以实现在数据库中更新数据集中修改的内容。

(1)　返回数据集。

在 Web 服务中使用数据集，和在单个应用程序中使用数据集的方式并不相同。在 Web 服务中，需要在服务器端获取数据集以后，把数据集对象通过 SOAP 消息发送到客户端。

由于数据集是一个.NET 对象，可以直接在.NET 内被序列化，所以我们不用做过多的操作，直接在 Web 服务方法中返回数据集对象即可。

(2)　使用数据集更新数据库。

前面我们说过，数据集还可以跟踪修改，而且可以使用数据集来更新数据库中的数据，所以还可以把客户端操作过的数据集对象发送到服务器中，使用数据集对数据库进行修改操作。

也就是说，我们可以在 Web 服务方法中声明一个 DataSet 参数，接收从客户端提交过来的数据集对象，使用该数据集对象更新数据库。



在使用数据集更新数据库时需要注意：客户端发送回来的数据集中带有从数据库中检索得到的所有初始数据，以及客户端对数据集的修改操作，所以我们需要尽量保持数据集中最必要的数据，否则会大大增加网络传输的数据量，降低程序效率。

2. 二进制数据

在网络中，文件的传递一直是非常让开发人员头疼的问题。如果 Web 服务中允许在客户端和服务器端之间双向地传递二进制数据，那么就可以很方便地实现文件的传递。但是 Web 服务中使用 XML 文本的形式封装消息，所以无法在 Web 服务中直接传递二进制对象。

因为在.NET Framework 中，可以使用内存流(Memory Stream)读取文件并将其转换为字节数组，所以可以把文件以字节数组的方式在网络中传递。

我们知道在 Web 服务中传递一个二进制数组是非常容易的事情。

当然，在 Web 服务的客户端，接收到二进制数组以后，使用内存流进行加工，就可以还原服务器发送回来的文件。

6.3.2　实例描述

使用 Web 服务在网络中传输文件是非常惬意的事情。我们可以将二进制的文件转换为可以被 XML 序列化的字节数组，然后分别在客户端和服务器端操作这个字节数组。

本实例，我们就使用 Web 服务来实现一个上传和下载图片的功能。需要首先在 Windows 窗体应用程序的客户端选择一个图片文件，并上传到服务器。然后在客户端将刚才上传的这个图片文件下载下来，显示到窗体中。

6.3.3　实例应用

【例 6-1】 使用 Web 服务上传和下载图片

(1) 在 Web 服务器端创建一个 Web 服务，命名为 WSUpload。

(2) 在 WSUpload 中添加一个上传文件的 Web 方法，命名为 UploadFile。该方法接收一个字节数组和一个文件名称，使用文件流将字节数组保存到磁盘中的指定文件中，代码如下：

```
[WebMethod]
public void UploadFile(byte[] content, string fileName)
{
    fileName = "D:\\" + fileName;

    //创建一个文件流对象，并初始化
    FileStream fs = new FileStream(fileName, FileMode.OpenOrCreate);
```

```
    //向文件流中写入内容
    fs.Write(content, 0, content.Length);

    //关闭流
    fs.Close();
}
```

(3) 在 WSUpload 中添加一个下载文件的 Web 方法，命名为 GetFile。该方法接收一个文件名，实现把磁盘中的指定文件序列化成一个字节数组，并发送到客户端，代码如下：

```
[WebMethod]
public byte[] GetFile(string fileName)
{
    fileName = "D:\\" + fileName;

    //创建一个文件流对象，并初始化
    FileStream fs = new FileStream(fileName, FileMode.Open);

    //创建一个二进制数组
    byte[] bs = new byte[fs.Length];

    //从文件流中读取内容
    fs.Read(bs, 0, bs.Length);

    //关闭流
    fs.Close();

    //以二进制数组的形式返回文件内容
    return bs;
}
```

(4) 在客户端 Windows 应用程序中添加对 Web 服务 WSUpload 的服务引用。

(5) 创建一个用于上传和下载图片文件的窗体，设计界面如图 6-4 所示。

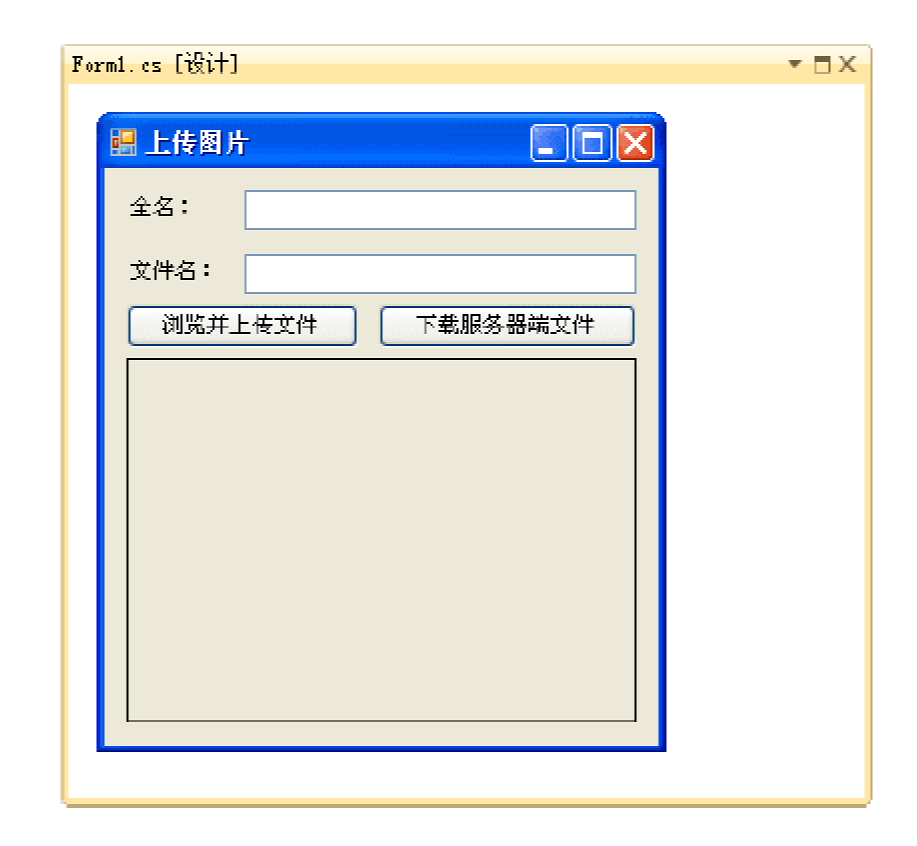


图 6-4 【上传图片】窗口设计界面

(6) 在【浏览并上传文件】按钮的单击事件中使用一个打开文件对话框(OpenFileDialog)浏览本地磁盘中的一个文件，并将其序列化为一个字符数组。然后创建一个上传文件的 Web 服务的代理对象，实现上传文件。代码如下：

```
private void btnSearch_Click(object sender, EventArgs e)
{
    //创建一个文件浏览对话框，并弹出该对话框用于获取一个本地磁盘中的文件名称
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.ShowDialog();
    string fileName = string.Empty,
        shortFileName = string.Empty;
    try
    {
        //获取文件浏览对话框选中的文件名称
        fileName = ofd.FileName;
        shortFileName = fileName.Substring(fileName.LastIndexOf('\\') + 1);
        this.txtAllName.Text = fileName;
        this.txtFileName.Text = shortFileName;
    }
    catch
    {
        //如果在获取文件名称的过程中出现异常，捕获并使方法返回
        return;
    }

    //创建一个文件流对象，并初始化
    FileStream fs = new FileStream(fileName, FileMode.Open);
    //创建一个二进制数组
    byte[] bs = new byte[fs.Length];
    //从文件流中读取内容
    fs.Read(bs, 0, bs.Length);
    //关闭流
    fs.Close();

    //创建一个上传或下载文件的 Web 服务代理类对象
    ServiceUpload.WSUploadSoapClient upload = new
ServiceUpload.WSUploadSoapClient();
    //执行上传文件操作
    upload.UploadFile(bs, shortFileName);

    MessageBox.Show("图片上传成功！");
}
```

(7) 在【下载服务器端文件】按钮的单击事件处理程序中添加从 Web 服务中获取文件的代码。

这需要首先从 Web 服务器端获取指定文件的字符数组，并将其转换为位图对象，再显示到窗体中的 PictureBox 对象中，代码如下：

```
private void btnDownload_Click(object sender, EventArgs e)
{
    //获取文件名称
    string shortFileName = this.txtFileName.Text;
    //创建一个上传或下载文件的 Web 服务代理类对象
    ServiceUpload.WSUploadSoapClient upload = new
     ServiceUpload.WSUploadSoapClient();
    //upload.
    //获取以字节数组保存的文件内容
    byte[] bs = upload.GetFile(shortFileName);
    //创建一个内存流对象，并用字节数组初始化该对象
    MemoryStream ms = new MemoryStream(bs);
    //创建一个位图对象，并使用内存流对象初始化该位图对象
    Bitmap b = new Bitmap(ms);
    //将该位图对象显示到窗体中
    this.picImage.Image = b;
    //关闭流
    ms.Close();
}
```



在使用 Web 服务执行文件的上传或下载操作时，应用程序可能会因为上传的文件过大而抛出异常。因此需要在客户端和服务器端的配置文件中设置上传和下载的文件大小限制。

6.3.4 运行结果

首先运行 Web 服务项目，然后再运行客户端的 Windows 窗体应用程序“上传图片”。

在“上传图片”窗口中，单击“浏览并上传文件”按钮，选择一个本地磁盘中的图片文件(这里是 F 盘中的 ap.jpg 文件)，系统会自动上传该文件，在窗体上的文本框中显示这个图片的文件名，并且弹出对话框提示用户上传成功，如图 6-5 所示。

然后单击【上传图片】窗口中的【下载服务器端文件】按钮，系统会获取服务器端文件内容并显示到窗体中的图片框中，如图 6-6 所示。

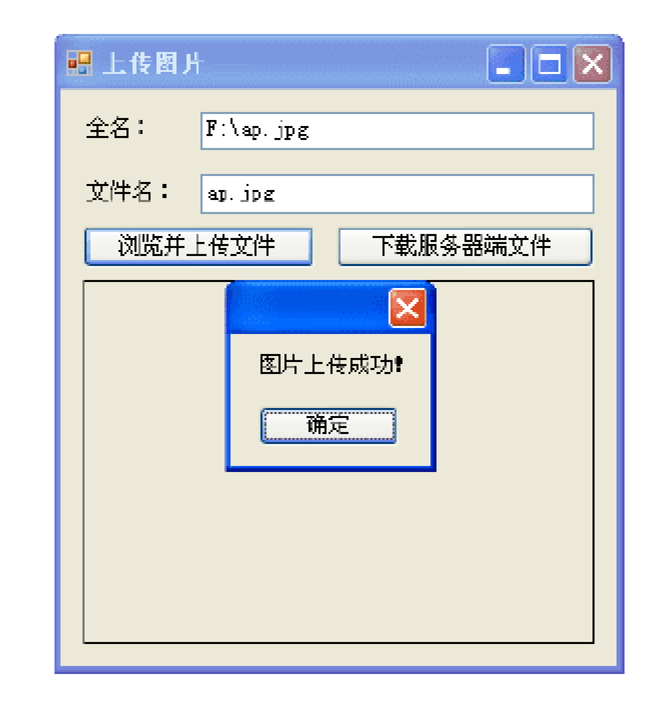


图 6-5　图片上传成功



图 6-6　图片下载成功

6.3.5　实例分析



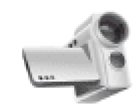
源码解析：

在本实例中，二进制文件和字符数组之间的转换操作执行了 4 次。首先在 Web 服务中编写一个接收上传文件功能的 Web 方法，接收客户端传递过来的字符数组，使用文件流将其保存到磁盘中；然后在 Web 服务中添加一个接收下载文件功能的 Web 方法，接收客户端传递的文件名称，从磁盘中读取该文件，并转换成字符数组，返回给客户端；接下来在客户端执行上传文件的操作时，使用文件流将本地文件转换为字符数组并上传到服务器。在执行下载操作时获取服务器端返回的字符数组，使用内存流将其转换为位图图像，并显示到窗体中。

6.4　隐藏用户的隐私信息

Web 服务一般都是在 Internet 中使用，然而 Internet 中的各种使用 Web 服务的平台和技术都不一定相同。因此，ASP.NET 所使用的默认的 SOAP 消息格式可能与其他技术的实现不兼容。所以为了应用程序的兼容性，我们就需要对 Web 服务使用的 SOAP 消息格式进行定制，以更好地与其他平台上的应用程序进行通信。

在 ASP.NET Web 服务中，.NET Framework 提供了一些格式化 SOAP 消息格式的方法，我们可以很方便地在代码中指定 SOAP 消息的格式。



视频教学：光盘/videos/06/隐藏隐私信息.avi　　　　长度：23 分钟

6.4.1　基础知识——定制 SOAP 消息

SOAP 消息一般具有两种编码格式：Document 格式和 RPC 格式。在.NET Framework 中对 SOAP 消息进行编码的方法也有两种：使用属性定制 XML 和使用 XML 序列化 SOAP 消息。

下面来分析 SOAP 的消息编码格式，并了解这两种编码 SOAP 消息的方法。

1．消息的编码格式

SOAP 消息有两种编码格式：Document 格式和 RPC 格式，我们可以用这两种编码格式来指定如何对整个 Body 元素进行编码。

当使用 Document 样式时，系统将基于 XML 架构定义来格式化 Body 元素中的内容；使用 RPC 格式时，将会把所有的参数信息放到一个单一的元素中，然后再把这个元素放到 Body 元素中。在使用 RPC 格式时，这个单一的元素名称与 RPC 方法的名称相同，其中的子元素对应于该方法的参数和返回值。

对于单独参数的编码，我们可以使用 Literal 格式或 Encoded 格式。在 Literal 格式中，每个参数都直接保存在一个与之对应的元素中；在 Encoded 格式中，将会为每个参数定义一个不

同的复杂类型，然后把参数保存在这个复杂类型的元素中。

在 WSDL 规范中，soap:body 扩充性元素包含一个 Use 属性，这个属性就是用来指定单个参数的编码格式的。当整个 SOAP 消息使用 Document 格式进行编码时，可以设置单个参数的编码格式使用 Literal 或 Encoded。当整个 SOAP 消息使用 RPC 格式进行编码时，还可以将单个参数使用 Encoded 编码格式。

2．使用属性定制 XML

.NET Framework 中提供了 4 个属性用于设置 SOAP 消息的编码格式，分别是：SoapDocumentMethodAttribute、SoapDocumentServiceAttribute、SoapRpcMethodAttribute 和 SoapRpcServiceAttribute。

其中 SoapDocumentServiceAttribute 属性和 SoapRpcServiceAttribute 属性可以用来设置整个 Web 服务类的默认编码格式。在没有专门指定编码格式的方法中将使用这些属性设置的编码格式。

剩下的 SoapDocumentMethodAttribute 属性和 SoapRpcMethodAttribute 属性可以用来设置特定方法的编码格式，这些属性设置将会覆盖默认的编码样式。

1） SoapDocumentServiceAttribute 属性

SoapDocumentServiceAttribute 属性需要在 Web 服务的实现类上使用。它可以把该类中所有 Web 方法的 SOAP 消息编码设置为 Document。如下所示：

```
//其他配置省略
[SoapDocumentServiceAttribute]
public class Default : System.Web.Services.WebService
{
    //代码省略
}
```

在使用 SoapDocumentServiceAttribute 属性时，还可以通过使用 Use 属性指定参数的编码格式。Use 属性是一个枚举类型 SoapBindingUse 的值，它有三个可选值，分别是 Default、Encoded 和 Literal。关于这三个可选值的说明如表 6-2 所示。

表 6-2 Use 属性的可选值

值	说 明
Default	为相应的 XML use 属性指定空字符串("")值
Encoded	使用给定的编码规则对消息部分进行编码
Literal	消息部分表示具体架构

另外，SoapDocumentServiceAttribute 属性中还有一个 ParameterStyle 属性，它用来指定是直接把参数放到 Body 元素中还是先把它放到一个 XML 元素中，然后再放到 Body 元素中。ParameterStyle 属性是一个枚举类型 SoapParameterStyle 的值，它也有三个可选的值，分别是 Default、Bare 和 Wrapped。关于这三个可选值的说明如表 6-3 所示。

表 6-3　ParameterStyle 属性的可选值

值	说　明
Default	指定使用 Web 服务的默认参数格式
Bare	将 Web 服务的参数放在 SOAP 请求或 SOAP 响应的 Body 元素之后
Wrapped	将 Web 服务的参数封装在 SOAP 请求或 SOAP 响应的单个 XML 元素中。

2)　SoapRpcServiceAttribute 属性

该属性用来将整个 Web 服务的 SOAP 消息的默认编码设置为 RPC 格式。设置格式如下所示：

```
//其他配置省略
[SoapRpcServiceAttribute]
public class Default : System.Web.Services.WebService
{
    //代码省略
}
```

3)　SoapDocumentMethodAttribute 属性

该属性用来为特定的方法指定 Document 编码格式。

SoapDocumentMethodAttribute 属性类包含以下可以影响 SOAP 消息格式的属性。

- Action：指定 SOAP 请求消息中的 SOAPAction 头字段，其默认值为 Web 服务的命名空间加上方法名。
- OneWay：指定客户端程序是否等待 Web 服务完成对所请求的方法的处理。如果这个属性设置为 true，则 Web 服务器在反序列化 SOAP 消息以后，调用被请求方法之前就会向客户程序返回一个 HTTP202 的状态码，表示它已经接受这个请求。但是在这种情况下，客户端就不会接收到带返回结果的方法调用的返回值，而是一个 HTTP 202 的响应消息。如果该属性值为 false(默认值)，表示客户端程序必须等待 Web 服务完成调用方法的处理。此时客户端会接收到一个 HTTP 200 的响应消息，表示执行成功，并且同时会得到方法的执行结果。
- ParameterStyle：指定是否把方法的参数封装到 Body 元素下的一个单一元素中。该属性的取值与效果同 SoapDocumentServiceAttribute 属性中的 ParameterStyle 属性完全一样。
- RequestElementName：指定 SOAP 请求消息中与该方法对应的元素的名称。在默认情况下，这个元素的名称与方法名相同。



当 ParameterStyle 属性的值为 Bare 时，方法的参数直接出现在 Body 元素中，所以在这种情况下，RequestElementName 属性的值不起作用。

- RequestNamespace：指定该方法对应的 SOAP 请求消息的命名空间，默认值为 Web 服务的命名空间。
- ResponseElementName：指定 SOAP 响应消息中与该方法对应的元素的名称。默认情况下，方法对应的元素的名称为方法名加上 Response 后缀。
- ResponseNamespace：指定 SOAP 响应消息中该方法所属的命名空间。默认情况下，命名空间为 Web 服务的命名空间。
- Use：指定方法的参数在 Body 元素中的格式化方式。它的取值和作用同 SoapDocumentServiceAttribute 属性中的 Use 属性完全相同。

4) SoapRpcMethodAttribute 属性

该属性为特定的方法指定 RPC 编码格式。

SoapRpcMethodAttribute 属性除了没有 Use 属性和 ParameterStyle 属性以外，其他的属性与 SoapDocumentMethodAttribute 属性完全相同。

由于在 RPC 格式下将使用 Message 元素和 portType 元素来格式化方法对应的元素，而不是使用 types 元素中定义的类型，所以 RequestElementName 属性中指定的值将不会对 SOAP 消息产生任何影响。

3．使用 XML 序列化 SOAP 消息

序列化是将对象转换成容易存储或在网络中进行传输的数据的过程。反序列化和序列化刚好相反，是将序列化过的数据重新整理并构造成对象的过程。

使用 XML 序列化就是将对象转换成 XML 流。XML 序列化只转换对象的公共字段和属性，并不转换方法、索引器、私有字段和只读属性。

因为 SOAP 消息就是使用 XML 序列化生成的数据，所以我们还可以通过控制 XML 序列化的操作来定制最终生成的 SOAP 消息。

对 XML 序列化过程的控制主要通过指定属性来完成。

1) 使用属性控制 XML 序列化

我们可以在类、类的公有字段、类的属性、方法的参数和方法的返回值上应用一些属性，用来指定它们在 XML 文档中对应元素的数据类型、名称和命名空间等信息。

例如，有一个名为 Book 的类，其中有两个公有字段 Name 和 ISBN。如果想要为 Name 字段指定一个不同的元素名称，就可以使用 XmlElement 属性为类或成员指定一个别名 BookName，如下所示：

```
public class Book
{
    [XmlElement(ElementName = "BookName")]
    public string Name;
    public string ISBN;
}
```

这样在使用 XML 序列化时，公有字段 Name 对应的元素名称就该是 BookName，而不是 Name 了。


类 XML 在 System.Xml.Serialization 命名空间中声明，所以在使用 XmlElement 之前不要忘了引入命名空间。

下面，再来了解一些比较常用的配置 XML 序列化的属性。

- XmlAttribute：用户可以在公共字段、属性、方法的参数和返回值上使用该属性来序列化一个 XML 属性。
- XmlElement：用户可以在公共字段、属性、方法的参数和返回值上使用该属性来序列化一个 XML 元素。
- XmlIgnoreAttribute：用户可以在类的公共属性和字段上使用这个属性来指定当序列化时需要排除的字段。使用该属性声明的字段将不被序列化。
- XmlRootAttribute：用户可以在公共类的声明中使用这个属性来指定把该类序列化为 XML 文档的顶级元素(即根元素)。
- XmlTypeAttribute：用户可以在公共类的声明中使用这个属性来指定它所对应的 XML 类型的名称和命名空间。

2)　Web 服务的 XML 序列化

.NET Framework 为定制 Encoded 编码格式的 SOAP 消息提供了专门的属性，这些属性以 Soap 开头，它们可以改变采用 Encoded 格式编码的 XML 序列化结果。

下面来看一下常用的控制 Encoded 格式 SOAP 消息的序列化结果的属性。

- SoapAttribute：用户可以在公共字段、属性、参数或返回值上使用该属性来序列化一个 XML 属性。
- SoapElement：用户可以在公共字段、属性、参数或返回值上使用该属性来序列化一个 XML 元素。
- SoapIgnore：用户可以在公共字段和属性上使用这个属性来指定在序列化时需要排除的项。即在序列化时跳过这些项。
- SoapType：用户可以在公共类的声明中使用这个属性来指定把该类序列化为 XML 类型，并可以同时指定类型的名称和所属的命名空间。

6.4.2　实例描述

在本例学习定制 SOAP 消息的方法时，我突然想到前段时间参与设计的一个项目中曾经使用过类似的功能。

这个功能的需求是这样的：在一个企业内部的信息管理系统中，需要向外发布一个接口，可以让关系密切的客户通过 Web 服务来查询用户信息。但是在这些用户信息中，有一步分信息是用户个人的隐私信息，这是不可以随意向外公布的(比如用户的个人联系方式)。所以在设计这个 Web 服务时，需要屏蔽这些不需要公布的个人信息。

对于这个功能，使用属性控制 XML 序列化的内容，就可以很方便地屏蔽这些相关的信息。下面来看这个实例。

6.4.3 实例应用

【例 6-2】 隐藏用户的隐私信息

(1) 准备一个用于封装用户信息的实体类 UserInfo。该类的结构如下所示：

```
public class UserInfo
{
    private string name = string.Empty;
    private int age = 0;
    private string email = string.Empty;
    private string phone = string.Empty;
    private string mobile = string.Empty;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    public int Age
    {
        get { return age; }
        set { age = value; }
    }
    public string Email
    {
        get { return email; }
        set { email = value; }
    }
    [XmlIgnore]
    public string Phone
    {
        get { return phone; }
        set { phone = value; }
    }
    [XmlIgnore]
    public string Mobile
    {
        get { return mobile; }
        set { mobile = value; }
    }
}
```

在这里要实现在 Web 服务中屏蔽用户的电话号码(Phone 属性)和手机号码(Mobile 属性)这两项信息，所以在实体类中使用 XmlIgnore 属性声明这两个类属性为 XML 序列化的排除项。这样就实现了不序列化这两项，当然在 Web 服务中传递的时候也不会传递这两项的内容了。

(2) 创建一个 Web 服务类，命名为 WSUsers。

(3) 在 Web 服务 WSUsers 中声明一个用于获取用户信息的 Web 方法，名为 GetUser，添加相应的业务逻辑代码，如下所示：

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
public class WSUsers : System.Web.Services.WebService
{
    [WebMethod]
    public UserInfo GetUser()
    {
        return new UserInfo()
        {
            Name = "张浩华",
            Age = 24,
            Email = "joke.he@163.com",
            Mobile = "13888888888",
            Phone = "0371-99999999"
        };
    }
}
```

这里我们模拟查询操作，不接受条件，只向用户返回一个完整的用户信息。需要注意的是，这里初始化了 UserInfo 类中的所有属性，并返回给方法调用者。

6.4.4　运行结果

使用浏览器访问这个 Web 服务(URL：http://localhost: 2525/WSUsers.asmx)，如图 6-7 所示。

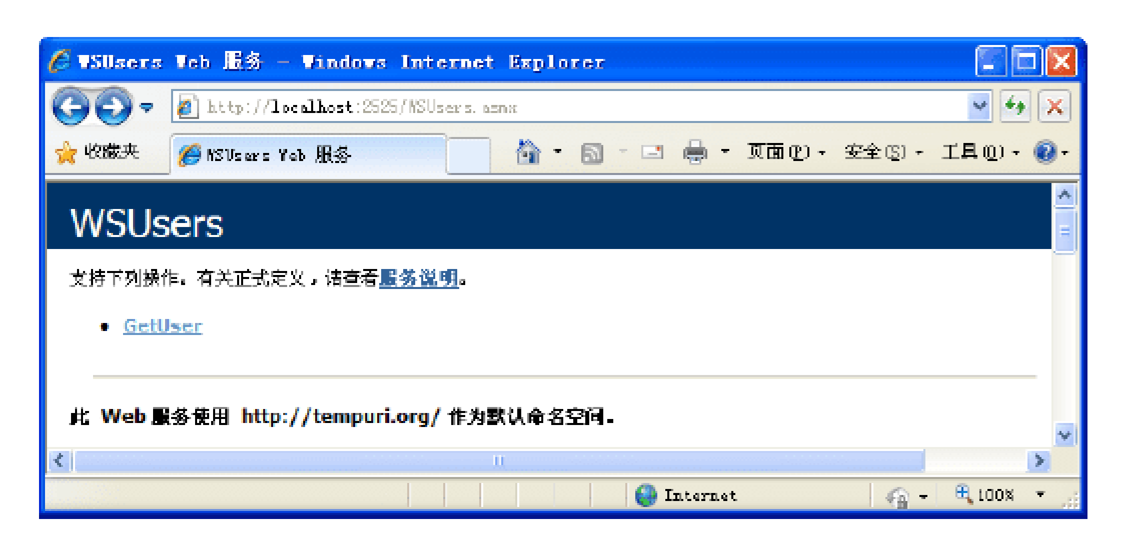


图 6-7　Web 服务 WSUsers 页面

在打开的页面中单击 GetUser 超链接，进入 Web 方法 GetUser 的页面，如图 6-8 所示。

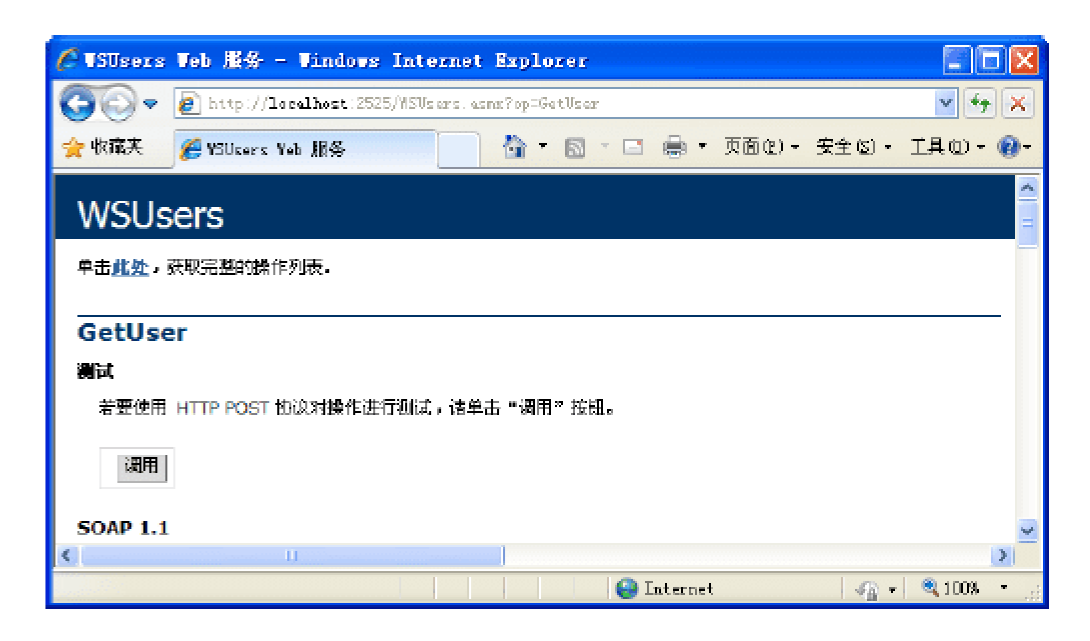


图 6-8　Web 方法 GetUser 页面

在该页面中单击【调用】按钮，浏览器会调用 Web 服务 WSUsers 的 GetUser 方法，并返回调用结果，如图 6-9 所示。

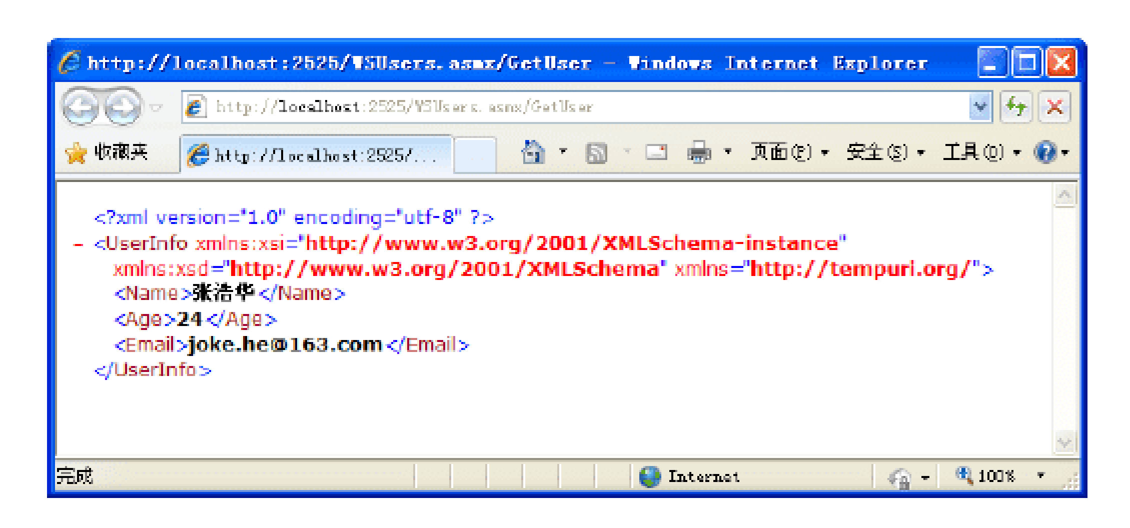


图 6-9　GetUser 方法执行结果

从执行结果中可以看到，Web 服务中虽然初始化了 UserInfo 类的所有属性，但是因为 Phone 属性和 Mobile 属性都声明为不序列化，所以这里只显示了剩余的三项内容。

6.4.5　实例分析



源码解析：

本实例实现在 Web 服务中隐藏用户的信息的功能，使用的是禁止序列化实体类的指定项的方法。

首先使用 System.Xml.Serialization 命名空间中的 XmlIgnore 类声明 UserInfo 类中不需要被 XML 序列化的类属性，使 Web 服务在序列化实体信息的时候跳过这些属性。然后在 Web 服务方法 GetUser 中创建一个 UserInfo 类的对象，初始化这个对象的所有属性，并返回给方法调用者。

6.5　记录客户端操作日志

SOAP 扩展是探索.NET Web 服务机制的一种方法。

迄今为止，我们使用的 SOAP 消息都是由 Web 服务自动完成的。比如解释 SOAP 请求和发送 SOAP 响应。我们所需做的就是提供一个 Web 方法，该方法使用从 SOAP 请求反序列化来的正确参数，同时被客户端调用。在方法执行完成后，发回一个仍旧包装在 SOAP 消息中的响应信息。整个过程不用人为干预，系统自动封装 SOAP 消息包。

.NET Framework 提供了一个对该过程进行控制的手段，这就是 SOAP 扩展。



视频教学：光盘/videos/06/记录客户操作日志.avi　　　　长度：32 分钟

6.5.1　基础知识——SOAP 扩展

客户端在访问 Web 服务时，需要把方法调用的信息和方法的返回结果序列化为能够在网

络中传输的 SOAP 消息，以便能够简单地实现 Web 服务的远程方法调用。整个操作的通信过程如图 6-10 所示。

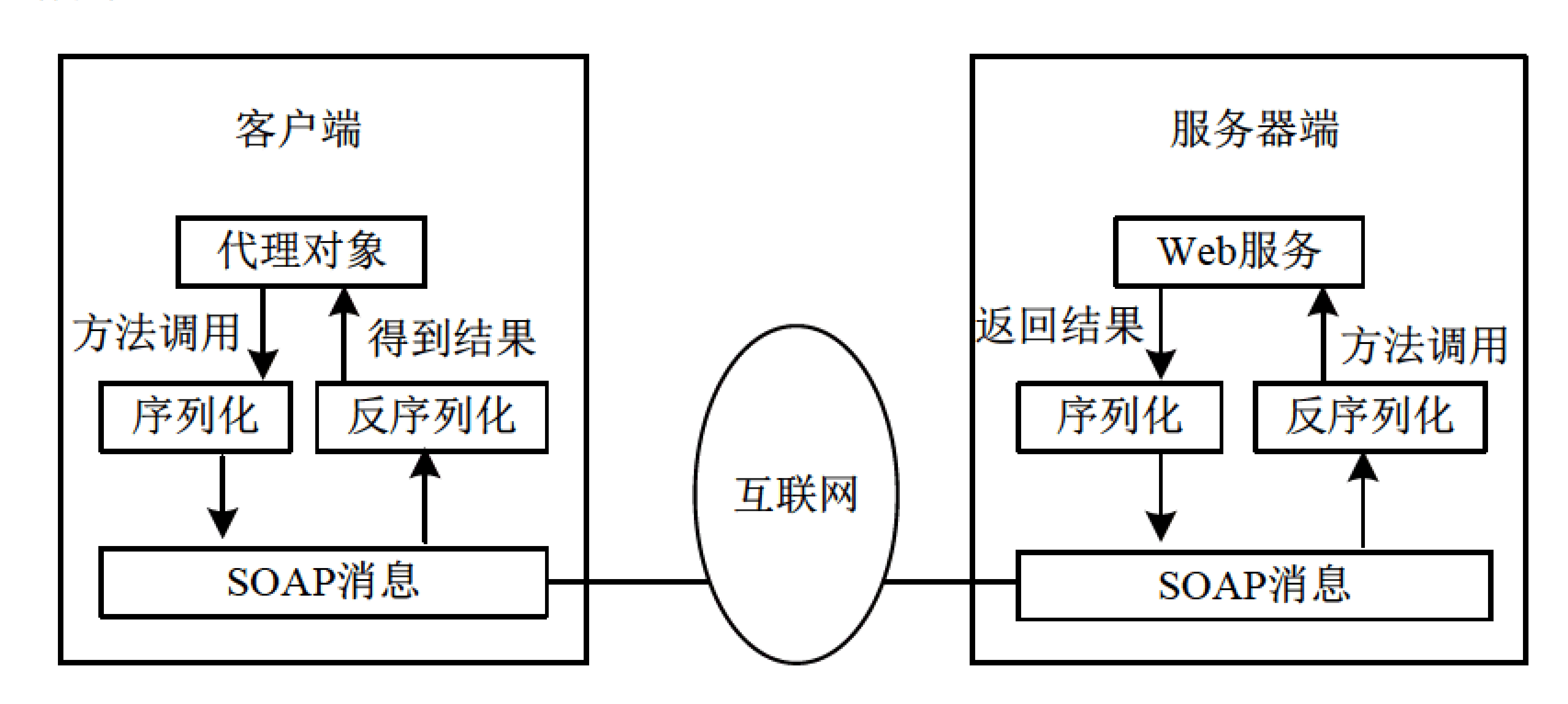


图 6-10　客户端和服务器之间的通信过程

从图 6-10 中可以看到客户端程序和 Web 服务之间的通信过程可以分为以下几步。

(1) 客户端创建代理对象并调用该代理对象的方法。
(2) 客户端根据代理对象的调用，将调用操作的相关信息序列化为 SOAP 请求消息。
(3) SOAP 消息通过 Internet 发送到 Web 服务器端。
(4) 服务器端把接收到的 SOAP 消息反序列化，得到相应信息。
(5) Web 服务器调用相应的 Web 服务方法，并返回方法执行结果。
(6) Web 服务器端将返回的结果序列化为 SOAP 消息，然后通知 Internet 响应给客户端。
(7) 客户端接收到服务器响应的 SOAP 消息，并进行反序列化。
(8) 代理对象得到反序列化的数据，返回给方法调用者。

1. SOAP 扩展工作原理

把对象转换为 XML 文本的过程称为序列化，把 XML 文本解析为对象的过程称为反序列化。在.NET Framework 中，完成这一转换功能的模块称为 XmlSerializer。

在 Web 服务中，使用 XML 序列化和反序列化一个数据对象的过程如图 6-11 所示。

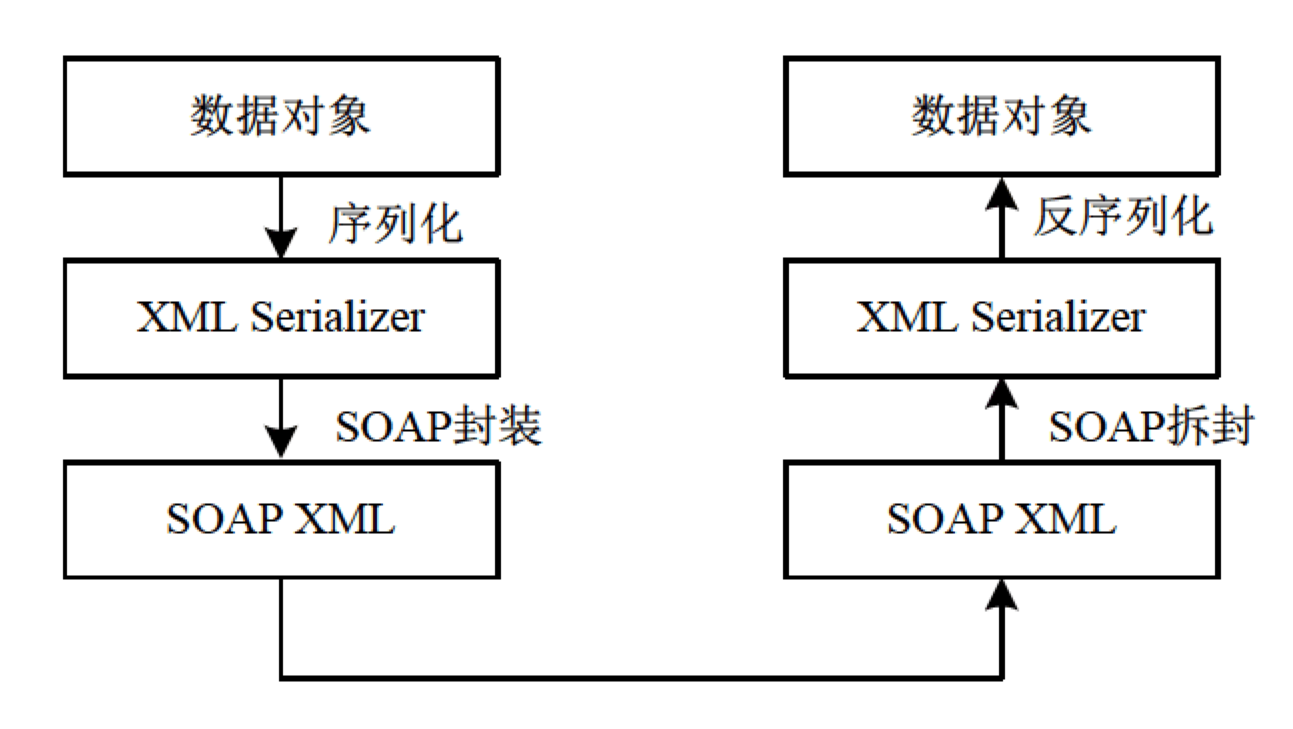


图 6-11　序列化和反序列化数据

SOAP 扩展允许用户在对象与 XML 转换的过程中访问数据。这意味着用户可以在 SOAP 流序列化数据前后访问它，然后再使用 Internet 发送给接收方。在服务器端接收到数据以后，

执行反序列化的过程中，SOAP 扩展也允许对数据进行访问。这意味着用户可以在反序列化SOAP 流的前后访问这些数据。

跟踪器扩展没有实现 SOAP 消息处理的全部工作。它只在消息序列化之前和反序列化之后查看消息，或者把 SOAP XML 消息写入磁盘中。

SOAP 扩展还可以修改 SOAP 消息的内容。比如可以在序列化之后加密消息内容，在反序列化之前解密消息内容。

2．创建 SOAP 扩展

.NET Framework 为使用 SOAP 扩展提供了很好的支持，它为用户提供了一个实现 SOAP 扩展的基类 SoapExtension，和一个用于 SOAP 扩展的辅助属性类 SoapExtensionAttribute。

在.NET Framework 中使用 SOAP 扩展需要从这两个类派生自己的扩展类和扩展属性类。其中扩展派生类可以用于提供所需的扩展操作，扩展属性派生类可以用来指定在哪些方法上使用扩展并且为扩展类提供一些信息。

使用 SOAP 扩展通常可以包含以下 3 个步骤。

(1) 从 SoapExtensionAttribute 类中派生一个类，并重写 SoapExtensionAttribute 类的一些属性。当然还可以添加一些自定义的属性。

(2) 从 SoapExtension 类派生一个扩展类并重写它的方法，以提供用户自定义的处理操作。

(3) 把派生的属性类应用到需要 SOAP 扩展的方法上，或者在 web.config 或 app.config 配置文件中添加扩展类引用。

通过上面这 3 个步骤，ASP.NETWeb 服务就知道在哪个方法上添加了 SOAP 扩展，并且可以使该方法自动调用指定的扩展操作。

1) 创建 SOAP 扩展属性类

SOAP 扩展属性是用来把特定的方法与用户自定义的 SOAP 扩展关联起来。如果要在某个特定方法上使用 SOAP 扩展，用户需要为其指定特定的 SOAP 扩展属性，然后 ASP.NET Web 服务就能够从这个属性中获取与该方法相关联的 SOAP 扩展类，并在适当的时候调用它。

创建 SOAP 扩展属性类，我们可以使用.NET Framework 中的 SoapExtensionAttribute 类，它将作为用户自定义扩展属性类的基类。

SoapExtensionAttribute 类位于命名空间 System.Web.Services.Protocols 中，它主要包括下面两个属性。

- ExtensionType：获取相关联 SOAP 扩展的类型。在派生类中必须重写这个属性，并通过它来返回特定 SOAP 扩展的类型。
- Priority：获取或设置 SOAP 扩展的优先级。我们在派生类中需要重写这个属性。因为可能同时存在多个 SOAP 扩展，所以需要使用优先级来确定它们的调用顺序。SOAP 扩展的优先级越高，它所处理的 SOAP 消息就越接近于网络中传送的 SOAP 消息。该属性接收大于或等于 0 的自然数，并且该属性值越小，表示该扩展的优先级越高(0 表示优先级最高)。

例如，要创建一个保存 Web 服务接收和发送的 SOAP 消息的 SOAP 扩展 SaveContent SoapExtension，那么就需要先为其创建一个扩展属性类 SaveContentSoapExtensionAttribute，以便使 Web 服务方法和 SOAP 扩展类关联起来，并设置一些必要的参数。

下面这段代码就是从 SoapExtensionAttribute 类中派生的一个新的属性类 SaveContent SoapExtensionAttribute：

```
[AttributeUsage(AttributeTargets.Method)]
public class SaveContentSoapExtensionAttribute: SoapExtensionAttribute
{
    public override Type ExtensionType
    {
        get { throw new NotImplementedException(); }
    }
    public override int Priority
    {
        get
        {
            throw new NotImplementedException();
        }
        set
        {
            throw new NotImplementedException();
        }
    }
}
```

从上面代码中可以看到，当自定义的类继承抽象类 SoapExtensionAttribute 以后，系统会要求必需重写该类中的两个属性 ExtensionType 和 Priority。



在上面代码中创建扩展属性类时，使用了 AttributeUsage 属性声明该扩展属性只能应用到方法成员上。

接下来在该类中重写基类的两个属性：ExtensionType 和 Priority，代码如下：

```
[AttributeUsage(AttributeTargets.Method)]
public class SaveContentSoapExtensionAttribute : SoapExtensionAttribute
{
    private int priority = 2;
    public override Type ExtensionType
    {
        get { return typeof(SaveContentSoapExtensionAttribute); }
    }
    public override int Priority
    {
        get
        {
            return priority;
        }
        set
        {
            priority = value;
        }
    }
}
```

至此，SOAP 扩展属性类 SaveContentSoapExtensionAttribute 就算定义完成了。

2) 创建 SOAP 扩展类

要使用 SOAP 扩展，还需要从 SoapExtension 类派生一个自定义扩展类。比如前面讲的 SaveContentSoapExtension 类，声明代码如下：

```
public class SaveContentSoapExtension : SoapExtension
{
   public override object GetInitializer(Type serviceType)
   {
      throw new NotImplementedException();
   }
   public override object GetInitializer(LogicalMethodInfo methodInfo,
    SoapExtensionAttribute attribute)
   {
      throw new NotImplementedException();
   }
   public override void Initialize(object initializer)
   {
      throw new NotImplementedException();
   }
   public override void ProcessMessage(SoapMessage message)
   {
      throw new NotImplementedException();
   }
}
```

从上面代码中可以看到，当自定义的类继承了 SoapExtension 类以后，自动生成了一些方法，说明这些方法对于 SOAP 扩展类来说是必需的。下面我们就来分别介绍这些方法。

(1) GetInitializer 方法。

在客户程序或 Web 服务执行期间，GetInitializer 方法只会被.NET Framework 调用一次，而且它的返回值将会被缓冲并作为参数传递到后续的 Initialize 方法调用。

GetInitializer 方法有两种重载形式，分别是：

```
public override object GetInitializer(Type serviceType)
```

和

```
public override object GetInitializer(LogicalMethodInfo methodInfo,
SoapExtensionAttribute attribute)
```

在 SOAP 扩展类中，我们需要同时重写 GetInitializer 方法的这两种重载。不过具体.NET Framework 最终会调用哪个重载形式，以及什么时候调用 GetInitializer 方法由 SOAP 扩展的指定方式确定。

如果通过在配置文件中添加引用的方式来使用 SOAP 扩展，则在第一次访问该配置文件范围内的所有 Web 服务时，.NET Framework 将调用 GetInitializer 方法的 GetInitializer (Type serviceType)重载形式。其参数类型为 Type，表示自定义扩展类的类型；返回值为 object，表示任意类型的对象。

如果用户通过在方法上添加属性的方式来使用 SOAP 扩展，则在第一次访问该方法时，.NET Framework 将调用 GetInitializer 方法的 GetInitializer(LogicalMethodInfo methodInfo, SoapExtensionAttribute attribute)重载形式。在调用 GetInitialize 方法时，通过 methodInfo 参数，可以获取应用 SOAP 扩展的方法的详细信息(比如方法的参数和返回值等)；通过 attribute 参数，则可以获取与 SOAP 扩展类对应的属性类的一些信息。

(2)　Initialize 方法。

在第一次使用 SOAP 扩展的 Web 服务方法时，.NET Framework 都会调用 Initialize 方法。系统会自动使用 GetInitializer 方法的返回值对象做为参数传递到这个方法中，用户可以使用这个信息来执行一些与方法相关的初始化操作。

(3)　ProcessMessage 方法。

该方法是 SOAP 扩展类的核心方法。在 SOAP 消息处理的每一个阶段，.NET Framework 都会调用这个方法，并且会向它传递一个表示 SOAP 消息的对象(SoapMessage 对象)。用户可以在这个方法中根据消息处理的不同阶段来对 SOAP 消息进行相应的处理。

如果 SOAP 扩展在客户端运行，则传递给这个方法的消息对象将是一个 SoapClientMessage 对象；如果 SOAP 扩展在服务器端运行，那么这个对象将是一个 SoapServerMessage 对象。

SoapMessage 对象表示特定阶段的 SOAP 请求或 SOAP 响应中的数据。其中包含一个 Stage 属性，可用于获取当前操作所处的阶段。

Stage 属性是一个 SoapMessageStage 类的枚举对象，用于指定 SOAP 消息的处理阶段，其可能出现的值有 4 个，如表 6-4 所示。

表 6-4　SoapMessageStage 枚举的值

值	说　明
BeforeSerialize	其值为 1。表示在序列化 System.Web.Services.Protocols.SoapMessage 之前的阶段
AfterSerialize	其值为 2。表示在序列化 System.Web.Services.Protocols.SoapMessage 之后，但在通过网络发送 SOAP 消息之前的阶段
BeforeDeserialize	其值为 4。表示在将 System.Web.Services.Protocols.SoapMessage 通过网络发送 SOAP 消息反序列化到对象之前的阶段
AfterDeserialize	其值为 8。表示在将 System.Web.Services.Protocols.SoapMessage 从 SOAP 消息反序列化到对象之后的阶段

上面所讲的这 4 个阶段可以运行在客户端或服务器端，客户端请求服务器的操作时无论哪个阶段都会调用 ProcessMessage 方法。图 6-12 所示为在不同阶段调用 ProcessMessage 方法的情况。

3．使用 SOAP 扩展

前面我们已经了解了如何创建 SOAP 扩展类和扩展属性类。但是这里只有两个类，我们还无法确定如何在 Web 服务中调用这个 SOAP 扩展。因此，还必须指定在哪个方法中使用 SOAP 扩展。

使用 SOAP 的扩展有两种方式：通过 SOAP 扩展属性来指定和通过配置文件来指定。

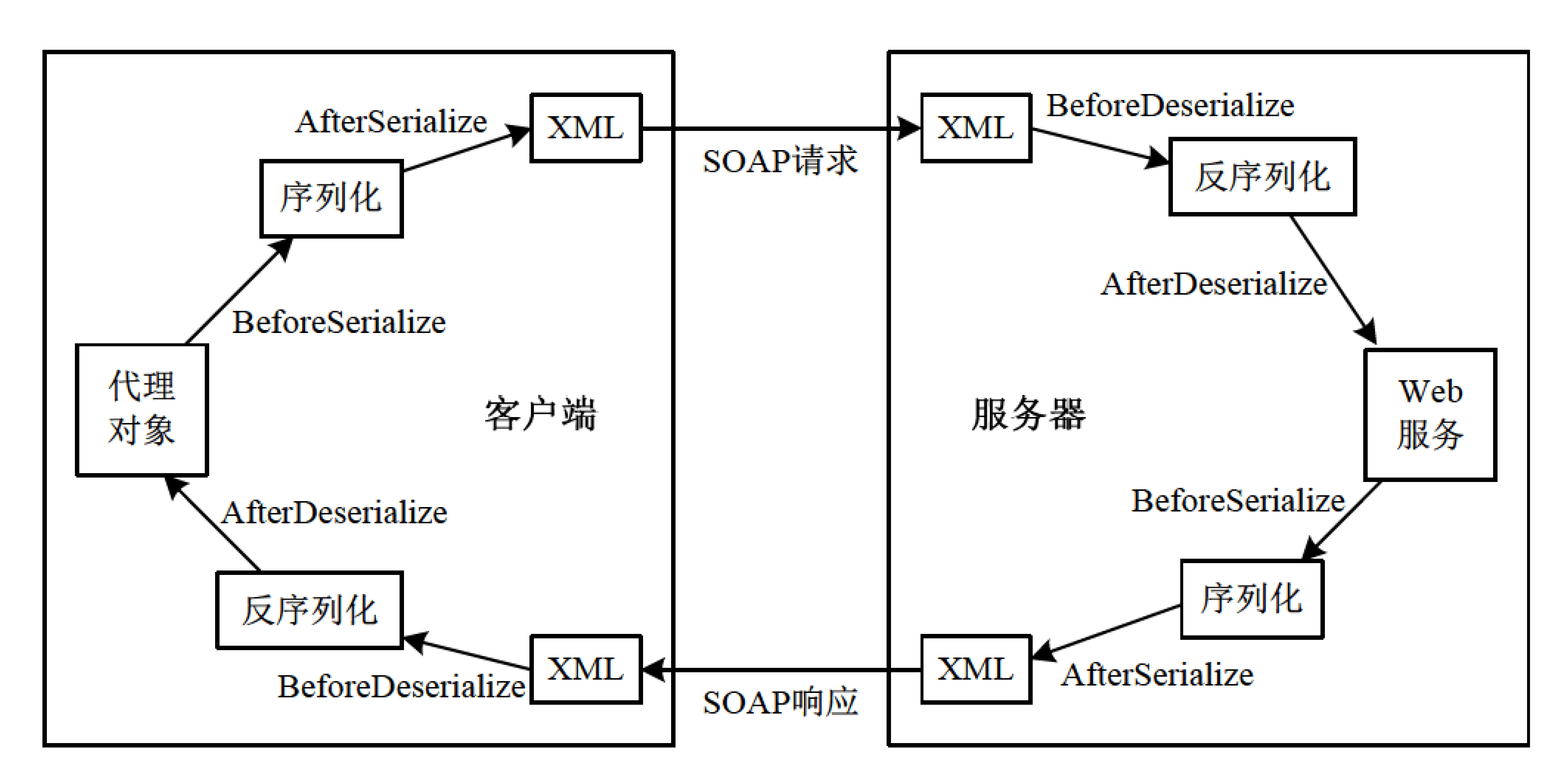


图 6-12　不同阶段调用 ProcessMessage 方法的情况

1)　使用 SOAP 扩展属性为方法指定 SOAP 扩展

使用 SOAP 扩展属性为方法指定 SOAP 扩展的方法非常简单，我们只需要在方法声明中指定前面所创建的 SOAP 扩展属性类(比如 SaveContentSoapExtension)，并设置一些参数就可以了，其他操作.NET Framework 会自动完成。

使用这种方式需要为每一个使用 SOAP 扩展的方法指定 SOAP 扩展属性。

比如在 Web 服务方法 HelloWorld 的声明中使用 SaveContentSoapExtension 属性指定其使用 SOAP 扩展，并把 SOAP 扩展的优先级设置为 1，代码如下所示：

```
[WebMethod]
[SaveContentSoapExtension(Priority = 1)]
public string HelloWorld()
{
    return "Hello World";
}
```

2)　使用配置文件指定 SOAP 扩展

使用配置文件指定 SOAP 扩展可以应用到该配置文件范围内的每个 Web 服务中的每一个方法上。使用配置文件的方法也比较简单，只需要在配置文件中添加一个 soapExtensionTypes 元素即可。


提示

在 Web 应用程序中，需要把 soapExtensionTypes 元素添加到 web.config 文件中；而在 Windows 应用程序中，则要把 soapExtensionTypes 元素添加到 app. config 中。

soapExtensionTypes 元素具有以下 3 个属性。

- type：该属性用来指定 SOAP 扩展的类型。在指定这个属性时，需要指定 SOAP 扩展类的全名(命名空间加上类名)以及包含它的程序集的名称。例如 SOAP 扩展位于 WebService 中，并且程序集名称也为 WebService，那么 type 属性的配置格式如下：type="WebService.SaveContentSoapExtension, WebService"。

- group：该属性用来指定该 SOAP 扩展所属的优先组。在 SOAP 扩展中总共有 3 个优先组，分别是使用扩展属性指定的 SOAP 扩展、在配置文件中指定的组 0 以及在配置文件中指定的组 1。
- priority：该属性用来指定 SOAP 扩展在组中的相对优先级。0 表示优先级最高。指定的值越大，优先级越低。

例如下面要在 Web 应用程序中的配置文件里添加一个 soapExtensionTypes 元素，指定 SOAP 扩展。代码如下：

```
<configuration>
   <system.web>
      <!-- 其他代码略 -->
      <webServices>
         <soapExtensionTypes>
            <add type="WebService.SaveContentSoapExtension, WebService"
                 group="1" priority="1" />
         </soapExtensionTypes>
      </webServices>
   </system.web>
</configuration>
```

在客户端添加 SOAP 扩展的过程与服务器端基本相同，都需要首先定义扩展类和扩展属性类，然后在代理类的方法上使用扩展属性指定相应的 SOAP 扩展，或者在配置文件中使用 soapExtensionTypes 元素指定 SOAP 扩展。

6.5.2　实例描述

SOAP 扩展可以让我们在 SOAP 工作的过程中对 SOAP 封装的消息包进行一些额外的操作。SOAP 扩展的用途有很多，可以使用它来记录请求、加密数据或压缩消息包。

本实例在前面实现过的“使用 Web 服务上传和下载图片”实例中，来添加一个 SOAP 扩展，用于记录客户端操作日志。

6.5.3　实例应用

【例 6-3】 记录客户端操作日志

(1)　在 Web 服务器端创建一个 SOAP 扩展类，命名为 SaveLogSoapExtension，用于记录日志，代码如下：

```
public class SaveLogSoapExtension : SoapExtension
{
   public override object GetInitializer(Type serviceType)
   {
      return null;
   }
```

```
    public override object GetInitializer(LogicalMethodInfo methodInfo,
     SoapExtensionAttribute attribute)
    {
        return null;
    }
    public override void Initialize(object initializer) { }
    public override void ProcessMessage(SoapMessage message)
    {
        switch (message.Stage)
        {
            case SoapMessageStage.AfterDeserialize:
                break;
            case SoapMessageStage.BeforeDeserialize:
                SaveLog(message);
                break;
            case SoapMessageStage.AfterSerialize:
                break;
            case SoapMessageStage.BeforeSerialize:
                break;
        }
    }
    private void SaveLog(SoapMessage message)
    {
        string filename = "D:\\log.txt";
        FileInfo file = new FileInfo(filename);
        if (!file.Exists) file.Create();
        using (FileStream fs = new FileStream(filename, FileMode.Append))
        {
            using (StreamWriter sw = new StreamWriter(fs))
            {
                sw.WriteLine("----------------------------------------");
                if (message.Stage == SoapMessageStage.BeforeDeserialize)
                {
                    Stream s = message.Stream;
                    StreamReader sr = new StreamReader(s);
                    sw.WriteLine(sr.ReadToEnd());
                    s.Position = 0;
                }
                sw.WriteLine(message.Action);
                sw.WriteLine(message.Stage);
                sw.WriteLine(DateTime.Now);
            }
        }
    }
}
```

(2) 添加一个扩展属性类 SaveLogSoapExtensionAttribute，代码如下：

```
[AttributeUsage(AttributeTargets.Method)]
public class SaveLogSoapExtensionAttribute : SoapExtensionAttribute
```

```
{
    private int priority = 1;

    public override Type ExtensionType
    {
        get { return typeof(SaveLogSoapExtension); }
    }
    public override int Priority
    {
        get
        {
            return priority;
        }
        set
        {
            priority = value;
        }
    }
}
```

这里的 Type 属性需要返回一个 SOAP 扩展的类。在这个 SOAP 属性类的前面使用 AttributeUsage 属性声明其为属性用例，并且声明该扩展属性只能使用在类上。

(3) 创建好 SOAP 扩展类和 SOAP 扩展属性类，下面在 Web 服务中使用它，代码如下：

```
public class WSUpload : System.Web.Services.WebService
{
    [WebMethod]
    [SaveLogSoapExtension]
    public byte[] GetFile(string fileName)
    {
        //方法中代码省略
    }

    [WebMethod]
    [SaveLogSoapExtension]
    public void UploadFile(byte[] content, string fileName)
    {
        //方法中代码省略
    }
}
```

6.5.4　运行结果

运行 Web 服务项目，然后使用 Windows 客户端再一次执行上传和下载图片操作，执行结果如图 6-13 所示(这里需要注意上传图片的文件名等信息，待会儿会在日志中出现)。

在这里向服务器上传了一个名为 3star.gif 的图片文件，然后请求服务器将其下载。经过这两个操作，服务器端应该记录了两条日志信息，并存储了请求的内容。

查看 SOAP 扩展记录的日志文件 D:\log.txt，内容如图 6-14 所示。



图 6-13　上传和下载文件

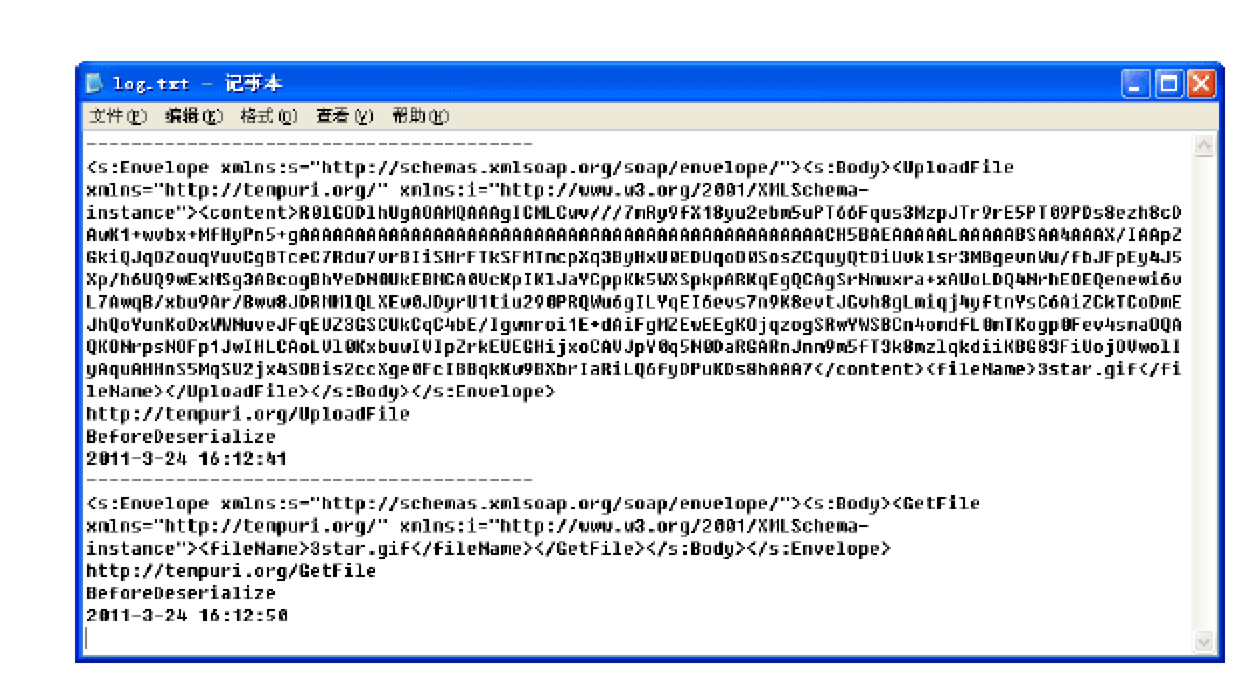


图 6-14　SOAP 扩展记录的日志信息

6.5.5　实例分析



源码解析：

本实例首先创建一个 SOAP 扩展类，在该类的 ProcessMessage 方法中判断 SOAP 扩展的执行状态，在 SOAP 消息反序列化之前获取 SOAP 请求的信息，记录用户请求的内容和其他相关信息。然后创建一个 SOAP 扩展属性类，在该属性类的 ExtensionType 属性中返回刚才创建的 SOAP 扩展类，并且在该 SOAP 扩展属性类前面使用 AttributeUsage 属性声明该属性只能应用于方法。最后，在 Web 服务 WSUpload 中的上传和下载文件的 Web 方法前分别使用创建的 SOAP 扩展属性类声明 SOAP 扩展。

6.6　常见问题解答

6.6.1　SOAP 概念的问题



SOAP 概念的问题。

网络课堂：http://bbs.itzcn.com/thread-15411-1-1.html

我知道 SOAP 是一种平台无关性的协议，也就是客户端和服务器端可以是不同的语言。但通过 SOAP 来实现传递信息，是一种什么机制呢？

【解决办法】

SOAP 只是一种传输协议。底层实现使用的是 XML+HTTP，其实它可以运行在任何文本传输协议之上。

给你看一个 SOAP 包的格式：

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
<!-- 头信息 -->
</soapenv:Header>
<soapenv:Body>
<!-- 消息主体 -->
</soapenv:Body>
</soapenv:Envelope>
```

协议这种东西只是一种支持应用的机制，离开实际的应用没有办法给你举例子。如果你想使用它，应该看具体的应用，比如 Web 服务。

6.6.2　SOAP 协议是否可以进行文件传输



SOAP 协议是否可以进行文件传输？
网络课堂：http://bbs.itzcn.com/thread-15412-1-1.html

SOAP 协议是否可以进行文件传输？

【解决办法】

SOAP 协议是一种文本传输协议，而文件是一种二进制数据，所以不能使用 SOAP 来直接传输文件。

不过只要对二进制文件进行一些转换，就可以使用 SOAP 协议传递了。

以 ASP.NET 为例，使用基于 SOAP 协议的 Web 服务传输文件，可以将二进制文件读入内存，并将其转换为一个字符数组，然后使用 SOAP 协议传输这个字符数组即可。

具体传递的文件大小，可以在 Web 服务的配置文件中进行限制。不过一般来说太大的文件不建议使用 SOAP 协议传输，否则将非常浪费应用程序性能。

6.7　习　　题

一、填空题

(1)　在一组完整的 SOAP 消息中，需要包含以下三个 XML 元素：__________元素、Header 元素和 Body 元素。

(2)　在服务器端通过 XML 序列化过的数据，需要在客户端执行__________操作以后才能被识别。

(3)　要实现一个 SOAP 扩展属性类，需要使自定义的类继承自__________类。

(4)　要实现一个 SOAP 扩展属性类，需要重写父类中的__________属性和 Priority 属性。

(5) SOAP 扩展类中的__________方法可用于处理 SOAP 消息执行的每一个阶段。

二、选择题

(1) 在标准的 SOAP 消息中，以下三个部分中__________是不必要的。

A. Envelope 元素　　B. Header 元素　　C. Body 元素

(2) 下面的各个数据类型中，__________不可以被 XML 序列化。

A. Date　　B. String　　C. List　　D. Stream

(3) 下列说法中，不正确的一项是__________。

A. SOAP 协议是一种轻量级的文本传输协议

B. SOAP 消息可以使用其他任何文本传输协议传输

C. SOAP 协议可以直接用来传输二进制数据

D. SOAP 使用 XML 来格式化数据

三、上机练习

上机练习 1： 实现一个文件上传工具。

前面讲过如何使用 SOAP 协议传输二进制文件。本实例我们就使用 Web 服务来实现一个简单的文件上传工具。

本实例需要实现浏览本地文件，并上传选中的文件的功能。

具体要求：用户在界面中可以选择文件，选中一个文件后，将文件的完整路径显示在窗体中，然后用户可以选择上传该文件。用户上传后的文件名将以列表的形式显示在窗体中的列表控件中。执行效果如图 6-15 所示。



图 6-15　简单的文件上传工具



第 7 章　管理 Web 服务的状态

内容摘要：

在使用分布式应用程序时，我们要解决的最为重要的一个问题就是数据交互。当然，在不同的主机间进行数据传递，一个非常重要的问题就是主机间如何进行识别。

在 B/S 结构的应用程序中，所有的数据都在一个数据中心——Web 服务器中保存着。其他所有的客户端都将以“请求—响应”的模式请求 Web 服务器，以获得 Web 服务器中的数据，或者向 Web 服务器提交数据。

在 Internet 中，任何一个角落的计算机都可能对我们的 Web 服务器发送请求，而且可能不止一次发送请求。在设计 Web 应用程序时，我们不可能在同一个用户执行不同请求时都对该用户进行身份验证——那样将非常麻烦。但是，在用户的各个请求之间保留状态信息，却是可行的。

在 Web 服务器中，一般都会提供 Cookie、Application、Session 等对象，可以实现在客户端请求 Web 服务器时进行一些数据暂存。但是使用这些服务器对象时，要注意应用程序的设计问题。任何不合理的状态保存方案都将削弱大批量用户请求时的服务器性能。

本章我们就来详细了解一下在设计 Web 服务时，如何对客户端的状态信息进行保持，如何对 Web 服务的状态信息进行管理。

学习目标：

- 了解 ASP.NET 中的状态管理
- 掌握会话状态的使用方法
- 掌握客户端 Cookie 状态的使用方法
- 掌握应用程序间的状态信息管理

7.1 Web 服务状态管理分析

随着近几年的发展，Web 应用程序成了企业应用程序设计变革的催化剂。随着 Web 应用程序的发展，我们必须保证应用程序在接收多个不同请求时保持状态，或者进行一些数据共享。

ASP.NET 在应用程序层和用户会话层两个方面为管理状态提供了一些服务端的元素，可以实现在 Web 应用程序中的不同请求之间进行状态保持的功能。

下面来了解一下 ASP.NET 为我们提供的这些状态保持的机制。

 **视频教学：光盘/videos/07/Web 服务状态管理分析.avi** **长度：7 分钟**

在设计一个 Web 服务时，需要考虑的重要问题就是：是否在请求之间保留关于客户和服务所完成工作的状态信息。如果状态保持设计的不合理，那么将会严重影响应用程序的性能。

1. 关于如何使用状态保持的问题

状态管理始终是程序开发中非常令人关注的一个话题，尤其是在使用 Web 应用程序的时候。

因为 Web 应用程序使用的 HTTP 协议是一个无状态协议，所以连接很少会超过几分钟，一般来说也就几秒钟而已。

在所有的 Web 应用程序中，服务器对用户的会话信息的保持一般有以下三种方式。

- 无会话状态方式：这种方式在用户请求以后，一旦断开连接，用户的状态信息自动失效。
- 在客户端保留信息：这种方式需要每次都将会话的状态信息通过网络发送到服务器上，服务器处理完以后将新的会话状态信息发送给客户端。这种方式不会额外占用服务器的内存，但是却会有较高的网络开销，增加了网络中的数据流量。
- 在服务器端保留信息：这种方式对于网络的开销很低，但是却需要额外占用服务器的内存资源。

上面三种方式中，如果只考虑应用程序的性能，第一种方式应该是最佳的解决方案。但是如果没有会话信息，在实现一些功能的时候就会非常困难，对于程序开发人员来说，这种方式并不可取。

对于程序开发人员而言，在服务器端保留状态信息的方法是最方便的。它不仅能简化代码，而且还可以像设计本地类方法一样设计 Web 服务方法。不过这种方式对内存的消耗较大。在有数千或数万个客户同时访问的时候，会话信息所占用的内存将是非常巨大的数字。

在客户端保存的信息比较持久，更能客观地体现用户的活动状态，而且也不会占用太多的服务器的内存。但是，如果在客户端保存的状态信息要在服务器端使用，就需要在客户端和服务器端之间互相传递。如果数据量大并且用户量也很大，对服务器的网络性能也是一个非常严峻的考验。

总之，现存的这些状态机制都存在一些缺点。我们如果要设计一个合理的 Web 应用程序，就需要详细地分析现实的状况，进行合理的处理。

2．ASP.NET 中关于状态管理的设计

在 ASP.NET 中，提供了一些状态管理的基础架构。使用诸如 Application、Session、Cookie 等服务器端对象，可以实现用户从一个 HTTP 请求到下一个 HTTP 请求的状态保持功能。

使用.NET Framework 实现的 Web 服务利用了 ASP.NET 的构造，所以和其他 ASP.NET 应用程序一样具有状态管理的功能。

ASP.NET 对象模式为 Web 服务中的状态管理保持了下面这些常用的核心功能。

- 通过使用 Cookie 或查询字符串，为保持用户的身份提供了一个机制。
- 通过使用 Session 和 Application 等固有的服务器端对象提供了一个保存用户的状态信息的机制。

在 ASP.NET 中，Session 对象和 Application 对象是我们在 Web 服务应用程序中管理状态时使用的最重要的技术。

Session 对象提供了保存单个会话的状态信息的功能，Application 对象提供了存储所有 Web 应用程序中的会话都可以访问的状态信息的功能。每个 Web 应用程序的用户都对应一个会话(Session 对象)，而所有的用户使用的 Application 对象是同一个。图 7-1 说明了 Web 应用程序的用户与会话和 Application 对象的关系。

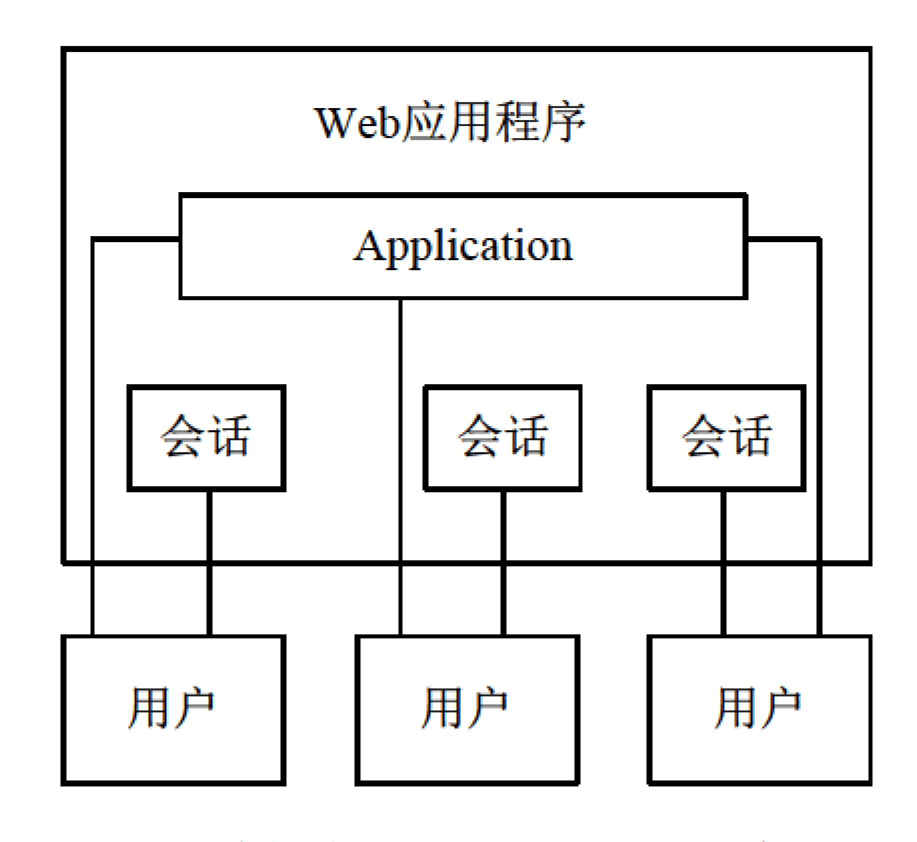


图 7-1　用户与会话和 Application 对象的关系

7.2　记录操作日志的简单计算器

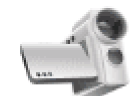
因为 Web 请求是一种无状态的请求。客户端在请求 Web 服务器以后，Web 服务器会对客户端的请求做出响应。响应结束后连接断开互相就不认识了，下一次请求就又是一次全新的请求和响应的过程。

不过客户端对一个 Web 服务器往往不会只做一次请求，所以客户端和服务器双方需要对多次的请求、响应状态进行保存，以备下次请求的时候使用。

在 ASP.NET 中，使用 Session 对象可以很方便地实现对每一个用户的状态信息进行保存，

以方便该用户在多次请求中重复使用保存的状态信息。

本节详细介绍 ASP.NET 是如何使用会话(Session 对象)记录会话状态信息的。



视频教学：光盘/videos/07/简单计算器.avi　　长度：23 分钟

7.2.1　基础知识——会话管理对象 Session

我们可以把“会话”理解为一个用户与 XML Web 服务之间的通信过程，一个会话过程中可以有多次“请求/响应”的过程。

就像两个人之间的谈话，每说一次话，对话的双方就完成了一次“请求/响应”。这样在完成无数次的“请求/响应”以后，也就是说很多句话以后，结束谈话才算一次会话过程结束。

简单地说，Web 应用程序中的会话，就是对应一个特定的用户对服务器连续请求的过程。对于这里的“用户”，在使用浏览器的时候是指当前浏览器；在请求 Web 服务的时候是指该 Web 服务客户端应用程序使用的代理类。

因为 HTTP 是无状态协议，所以无法知道不同的请求是否来自同一个用户。而如果我们把用户会话的状态信息保存起来就可以在多次“请求/响应”的过程中使用这些数据，以实现一些特定的功能。

例如，在设计网上商店系统的购物车功能时，可以将网上购物的过程分为几个步骤，即查找商品、挑选商品、管理购物车、结账等，如图 7-2 所示。

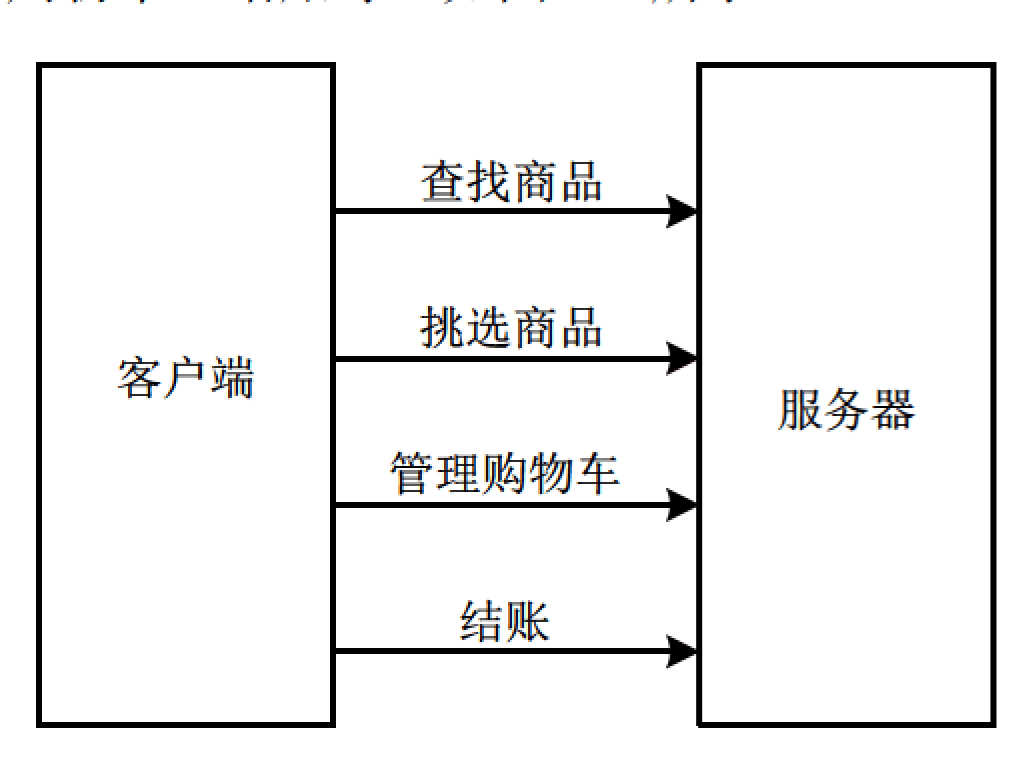


图 7-2　网上购物操作流程

在这多次请求当中，我们需要将用户多次请求的信息保存在这个会话中，以便下次请求时使用。

在 ASP.NET 中，对于客户端与服务器之间的会话信息的管理就需要使用 Session 对象。我们可以把每次请求中需要被下次请求访问的信息保存到 Session 中，以供下次请求使用。

因为一个会话与特定的用户相关联，所以需要使用一种方式将用户与服务器端的 Session 对象关联起来，以便能够在多个“请求/响应”过程中区分它们。每一个 Session 都是一个对象，每一个 Session 对象都有一个唯一标识，即 SessionID。客户端在请求 Web 服务器时，Web 服务器会在必要的时候为每一个请求创建一个 Session 对象，并分配一个 SessionID，这个 SessionID 就随 Cookie 发送到客户端。服务器在接收到客户端的请求以后，根据请求中的 Cookie

里保存的 SessionID 来区分每一个用户。



如果客户程序支持 Cookie，则 SessionID 将随着 Cookie 自动发送。否则，ID 将会作为请求的 URL 的一部分进行发送。

服务器端的 Session 对象其实就是一个类似于字典的东西，我们可以把 Session 对象看作一个字典或哈希表。

我们知道字典是以“键/值”对的形式保存数据的，所以可以使用以下方式将数据保存到 Session 中：

```
Session["Age"] = 24;
```

访问 Session 中的对象也可以通过指定的“键”来访问。例如用下面的代码可以从 Session 对象中读取刚才保存的值：

```
int age = (int)Session["Age"];
```

1. 初始化和释放会话状态

初始化 Session 对象的方法非常简单，直接使用 Session 对象为指定的键赋值即可。

Session 对象字典是一个 object 类型的字典，所以可以在其中保存任何类型的数据，而从 Session 中读取数据时，进行相应的类型转换即可。如下：

```
string name = "张小涛"
Session["Name"] = name;

if(null != Session["Name"])
{
    object oname = Session["Name"];
    string username = (String)oname;
}
```

前面说过，Session 是一个对象，那它是在什么时候创建的呢？这里我们只是直接存取，而没有使用 new 之类的关键字去创建它。其实，在一次会话过程中，第一次使用 Session 对象时，系统就自动创建了该 Session 对象。



如果在第一次使用 Session 对象或者用户的会话超时的时候，使用 Session 对象就必须进行非空判断，否则就会抛出异常信息。

既然使用 Session 对象时，需要保证这个 Session 对象的值不能为空，那么我们能不能在 Session 创建的时候自动初始化 Session 中的数据呢？答案当然是可以的。

所有的 ASP.NET Web 应用程序都可以包含一个名为 global.asax.cs 的文件，在这个文件中，我们可以添加应用程序启动和终止或会话创建和销毁的处理代码。

其中，处理会话创建的方法名为 Session_Start。当应用程序创建 Session 对象时，ASP.NET 都会调用这个方法。所以可以将初始化 Session 的代码添加到 Session_Start 方法中。这样每当启动新的会话时，系统将自动使用 Session_Start 方法中的代码初始化新创建的会话。

例如要在 Session 中初始化一个用户的登录名称为空字符串，可以这样修改 Session_Start

方法：

```
void Session_Start(object sender, EventArgs e)
{
    Session["LoginName"] = "";
}
```

2．在 Web 服务中使用会话状态

如果在 Web 服务中使用会话状态，则 Web 服务类必须要继承自 System.Web.Services.WebService 类。该类中声明了 Session、Application、Server、Context 等服务器端对象，类声明如下：

```
public class WebService : MarshalByValueComponent
{
    public WebService();

    public HttpApplicationState Application { get; }
    public HttpContext Context { get; }
    public HttpServerUtility Server { get; }
    public HttpSessionState Session { get; }
    public SoapProtocolVersion SoapVersion { get; }
    public IPrincipal User { get; }
}
```

在 Web 服务类继承自 System.Web.Services.WebService 类以后，还需要设置 Web 服务方法的 WebMethod 属性的 EnableSession 选项，将其值设置为 true，代码如下所示：

```
public class WebService1 : System.Web.Services.WebService
{
    [WebMethod(EnableSession = true)]
    public string HelloWorld()
    {
        return (string)Session["Hello"];
    }
}
```

3．配置会话状态

如果要在 ASP.NET 应用程序中管理和配置会话的属性，需要在应用程序配置文件中添加对 Session 的配置。配置 Session 需要在 web.config 文件中的 system.web 节点下面添加一个 sessionState 节点。

默认情况下，会话状态已经在.NET Framework 的配置文件中有相应的值。下面代码是在系统的.NET Framework 目录中的配置文件中的 Session 配置信息：

```
<sessionState mode="InProc"
    stateConnectionString="tcpip=loopback:42424"
    stateNetworkTimeout="10"
    sqlConnectionString="data source=localhost;Integrated Security=SSPI"
    sqlCommandTimeout="30"
```

```
    sqlConnectionRetryInterval="0"
    customProvider=""
    cookieless="UseCookies"
    cookieName="ASP.NET_SessionId"
    timeout="20"
    allowCustomSqlDatabase="false"
    compressionEnabled="false"
    regenerateExpiredSessionId="true"
    partitionResolverType=""
    useHostingIdentity="true"
    sessionIDManagerType="">
    <providers>
    <clear />
    </providers>
</sessionState>
```



虽然系统的配置文件中设置有默认的 Session 配置，但是如果有必要，我们可以在 Web 应用程序中添加一个 sessionState 节点并设置其相应的值。

从上面代码中可以看到，系统提供 Session 配置的属性非常多。但是在这些属性中，最常用的也只有 mode、stateConnectionString、sqlConnectionString、cookieless、timeout、stateNetworkTimout 等。关于这些配置属性，详细说明如表 7-1 所示。

表 7-1　sessionState 节点常用配置

名　称	描　述
mode	指定会话状态的存储位置，包括 Inproc、StateServer、SqlServer 和 Off 四个选项。其中 Off 选项用于禁用会话状态。对于剩余的三项，我们将在下文中详细介绍它们的功能
stateConnectionString	设置 Session 信息存储在状态服务中时使用的服务器名称和端口号。当 mode 属性的值为 StateServer 时，这个属性是必需的
sqlConnectionString	设置 Session 信息存储在 SQL Server 数据库中时使用的 SQL Server 连接字符串。当 mode 属性为 SQL Server 时，这个属性是必需的
Cookieless	指定是否不使用 Cookie 来标识会话。默认值为 UseCookies(使用 Cookie)。另外还可以设置为 true 或 false。如果不使用 Cookie，系统则自动将会话 ID 编码到 URL 中
Timeout	指定会话可以处于空间状态的时间，单位为分钟，默认值为 20，即会话空闲 20 分钟以后自动销毁
stateNetworkTimout	设置当使用 StateServer 模式存储 Session 状态时，经过多久空闲时间以后断开服务器与存储状态信息的服务器之间的 TCP/IP 连接。单位是秒，默认值是 10 秒钟

4．会话状态的存储方式

刚才我们了解了 ASP.NET 应用程序中的会话状态可以存储在不同的地方，下面进一步了解 ASP.NET 支持的这三种会话状态存储方式。

- Inproc：在 ASP.NET 工作进程(aspnet_wp.exe)中存储会话状态信息。这种方式存储的会话信息访问速度最快，但是却最不稳定。因为如果 ASP.NET 工作进程因为某种原因被终止，所有保存在其中的会话状态信息也将全部丢失。
- StateServer：在 ASP.NET 工作进程外的另一个单独进程(aspnet_state.exe)中存储会话状态信息。这种方式下会话状态由另外一个单独的进程来维护，而且该进程可位于另一台服务器上。这种方式的好处是会话状态不会受 ASP.NET 应用程序进程的影响，即使 ASP.NET 应用程序意外终止，会话状态也不会全部丢失。这种方式的另外一个好处是可以在多台服务器之间共享会话状态信息。
- SqlServer：在 SQL Server 数据库中存储会话状态信息。这种方式保存的会话信息非常稳定，但是访问速度最慢。因为会话信息保存在一个 SQL Server 数据库中，而不是在服务器内存中，所以即使服务器被关闭，会话状态也不会丢失，并且可以永久地保存下来。但由于每次请求都需要额外访问 SQL Server 数据库，所以运行速度也较前两种慢很多。

7.2.2 实例描述

Session 对象可以在服务器端为每一个访问服务器的用户保存一组状态信息。当然在一些使用 Web 服务的窗体应用程序中，我们也可能需要在一次会话中请求多次服务器。Web 服务器不可能每一次请求都对客户端进行一次验证，所以可以使用会话来保持窗体应用程序在 Web 服务器上的会话状态。

本实例，使用一个异步执行的计算器程序来演示在 Web 服务中的会话的用法。

7.2.3 实例应用

【例 7-1】 记录操作日志的简单计算器

(1) 创建一个执行计算操作的 Web 服务，名为 WSCounter。在这个 Web 服务类中，需要执行三个操作：一个加法操作、一个减法操作和一个登录用户的操作。

这里需要在执行加减法操作时记录是哪个用户执行的操作，所以要求使用 Session 保存用户登录的状态信息。

WSCounter 类的完整代码如下：

```
public class WSCounter : System.Web.Services.WebService
{
   [WebMethod(EnableSession = true)]
   public void Login(string username)
   {
      Session["User"] = username;
   }
   [WebMethod(EnableSession = true)]
   public Result Add(decimal v1, decimal v2)
   {
```

```
        Result result = new Result();
        if (null == Session["User"])
        {
            result.Message = "你没有合法权限";
            result.Value = 0;
        }
        else
        {
            result.Message = Session["User"] + " 于 " + DateTime.Now + " 执行了
             加法操作";
            result.Value = v1 + v2;
        }
        return result;
    }
    [WebMethod(EnableSession = true)]
    public Result Subtract(decimal v1, decimal v2)
    {
        Result result = new Result();
        if (null == Session["User"])
        {
            result.Message = "你没有合法权限";
            result.Value = 0;
        }
        else
        {
            result.Message = Session["User"] + " 于 " + DateTime.Now + " 执行了
             减法操作";
            result.Value = v1 - v2;
        }
        return result;
    }
}
```

(2) 这里用到一个封装执行结果的自定义类 Result，封装了执行结果和提示信息，代码如下：

```
public class Result
{
    public decimal Value { get; set; }
    public string Message { get; set; }
}
```

(3) 发布这个 Web 服务，然后在客户端应用程序中添加对该 Web 服务的 Web 引用。



客户端必须添加“Web 引用”，因为在调用 Web 服务时需要设置代理类的一些信息，所以使用“服务引用”不能实现该功能。

(4) 在窗体应用程序的计算器窗体中声明一个该 Web 服务的代理类变量，并在构造方法中初始化 CookieContainer 属性，代码如下：

```
public partial class Counter : Form
{
    localhost.WSCounter wsc = new localhost.WSCounter();
    public Counter()
    {
        wsc.CookieContainer = new System.Net.CookieContainer();
        InitializeComponent();
    }
    //其他代码略
}
```



因为窗体应用程序不是浏览器，所以代理类中并不能创建存储 Cookie 集合的对象，需要手动初始化 CookieContainer 属性。

(5) 下面来实现用户登录的功能。下面的代码接收用户在窗体中输入用户名，使用代理类调用 Web 服务执行登录功能，代码如下：

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string username = this.txtUsername.Text;
    if (username.Trim() != string.Empty)
    {
        wsc.Login(username);
        this.lstLog.Items.Add("用户 " + username + " 登录成功");
    }
}
```

(6) 实现执行加减法功能的方法，代码如下：

```
private void btnSub_Click(object sender, EventArgs e)
{
    decimal v1 = decimal.Parse(this.txtValue1.Text);
    decimal v2 = decimal.Parse(this.txtValue2.Text);
    var result = wsc.Subtract(v1, v2);
    this.lblResult.Text = result.Value.ToString();
    this.lstLog.Items.Add(result.Message);
}
private void btnAdd_Click(object sender, EventArgs e)
{
    decimal v1 = decimal.Parse(this.txtValue1.Text);
    decimal v2 = decimal.Parse(this.txtValue2.Text);
    var result = wsc.Add(v1, v2);
    this.lblResult.Text = result.Value.ToString();
    this.lstLog.Items.Add(result.Message);
}
```

在接收到用户输入的值以后，使用代理类对象调用 Web 服务，执行完成以后在窗体中显示执行结果。

7.2.4　运行结果

首先运行 Web 服务所在的 Web 应用程序。

然后，运行窗体应用程序，分别在 Num1 和 Num2 文本框中输入两个数，单击【减法】按钮，系统提示“你没有合法权限”，如图 7-3 所示。

在【用户名】文本框中输入 Joker，单击【登录】按钮。应用程序会使用该用户名向 Web 服务器执行登录，执行结果如图 7-4 所示。

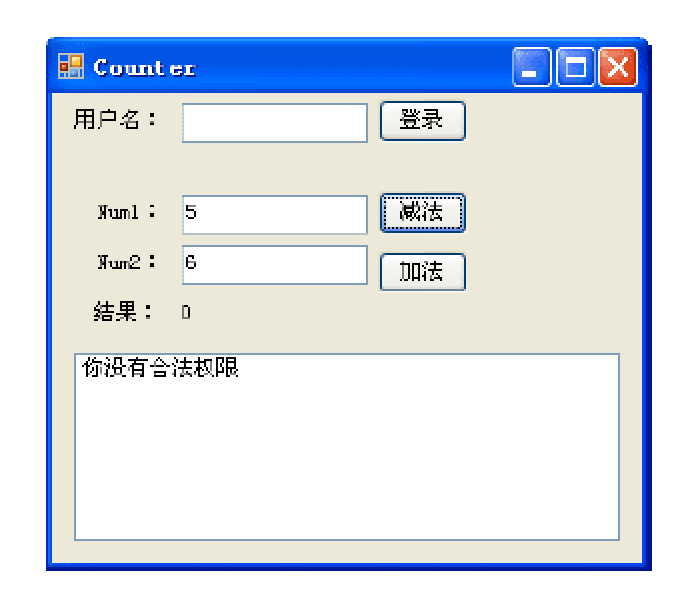


图 7-3　没有合法权限

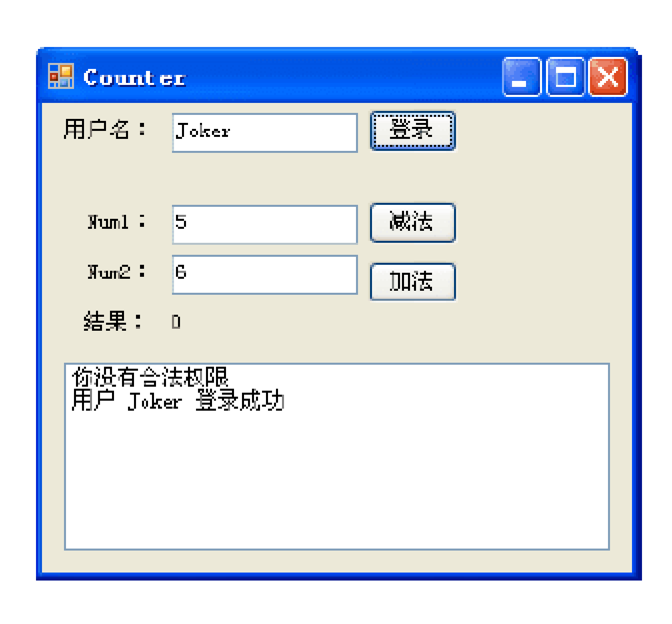


图 7-4　用户登录成功

登录成功以后，再单击【减法】按钮，窗体中将显示执行结果，如图 7-5 所示。

当然单击【加法】按钮一样可以得到相应的结果，如图 7-6 所示。

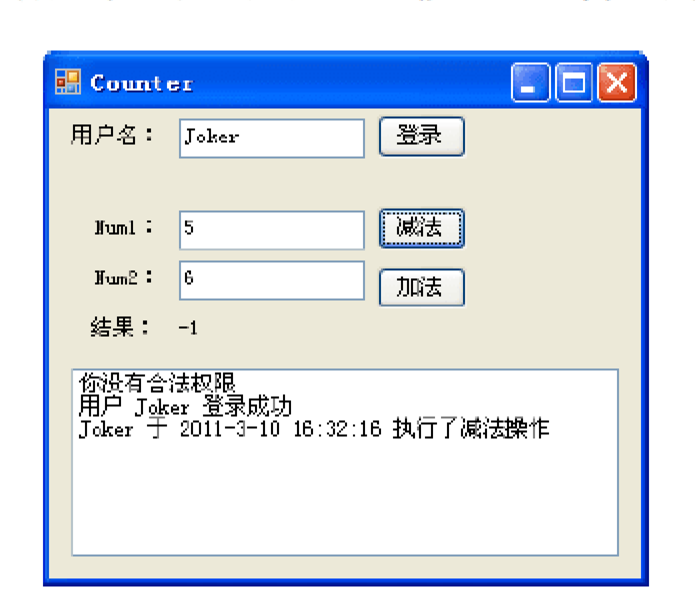


图 7-5　执行减法操作

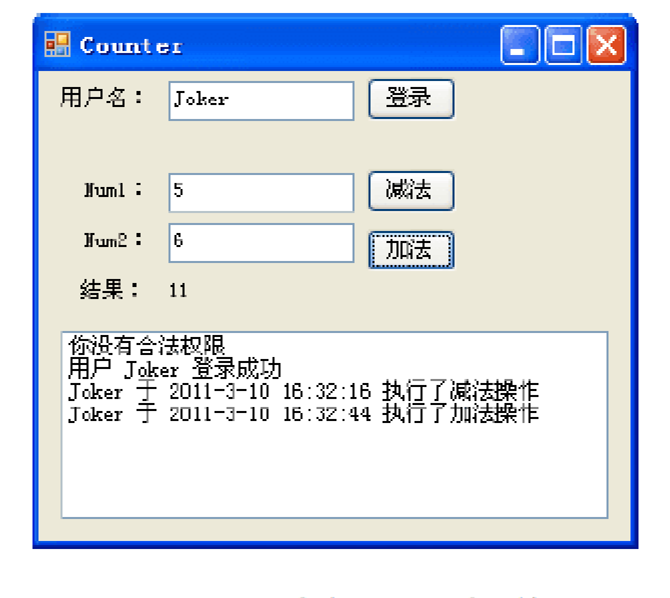


图 7-6　执行加法操作

7.2.5　实例分析



源码解析：

本实例中，首先在 WebService 中创建三个 Web 服务方法，分别执行登录、加法、减法操作。在执行登录操作时将用户提交的用户名保存到 Session 中，然后在加、减法操作中使用 Session 中的用户名组织相应的提示信息，接下来在客户端添加对该 Web 服务的 Web 引用。最后创建一个代理类对象，并设置一个 Cookie 集合。在窗体应用程序中分别使用这些方法进行相应的运算操作，并显示执行结果。

7.3　使用 Application 统计系统的在线人数

我们知道 Web 应用程序是多线程的，所以在设计 Web 系统时可以不考虑不同用户间的信息冲突问题。但是应用程序可能需要在不同的用户之间共享一些信息，或者多个用户共同维护一个数据信息。这个时候就需要考虑设计一个多个线程间都能访问一组信息的功能。

幸运的是，ASP.NET 提供了一个名为 Application 的服务器端对象，可以让我们在不同的用户间使用同一组共享的数据，达到数据共享的功能。

下面来详细了解 Application 对象是如何在 Web 服务中实现不同用户间的数据共享的。

 **视频教学：**光盘/videos/07/统计系统在线人数.avi　　长度：13 分钟

7.3.1　基础知识——应用程序对象 Application

会话状态可以保存单个用户的数据，而应用程序对象保存的是整个应用程序的全局数据。也就是说在应用程序对象中保存的数据可以在整个应用程序的范围内访问。

通常将一些应用程序内共享的少量数据保存在应用程序对象 Application 中。Application 中保存的数据总是在 ASP.NET 应用程序进程内部处理，不能在多个服务器之间共享或异地存储。另外，应用程序状态 Application 也不像 Session 一样，不需要使用 Cookie。

1．应用程序状态的使用

可以把 Application 对象看作存储全局信息的容器，它也是通过“键/值”对的形式来存储数据的。在 Web 服务中，Application 对象可以通过 WebService 类的 Application 属性来访问。

下面的代码演示了如何向 Application 对象中保存信息以及如何从 Application 中取出信息：

```
//保存信息
Application["Count"] = 20;
//取出信息
int count = (int)Application["Count"];
```



Application 对象和 Session 对象相同，是一个 object 类型的字典。所以我们在从 Application 对象中读取数据的时候需要进行相应的类型转换。

同会话状态 Session 相同，应用程序状态 Application 可以在应用程序全局文件 global.asax.cs 中进行初始化，也可以在访问时进行初始化。

与会话状态 Session 不同的是：应用程序的状态是全局性的，所以可以在应用程序启动的时候对它进行初始化。

在 global.asax.cs 文件中，应用程序启动的事件处理方法是 Application_Start，我们可以在该方法中编写代码初始化 Application 对象，代码如下：

```
void Application_Start(object sender, EventArgs e)
{
```

```
    Application["Count"] = 20;
}
```

2．解决使用应用程序状态时的多线程冲突

在使用应用程序对象时，需要注意的问题是应用程序对象的访问冲突。

因为应用程序状态是全局性的，可以被多个用户访问，所以在同一时刻，可能会有多个用户在访问它，这可能会造成应用程序对象的访问冲突问题。

例如要实现一个访问量统计的功能，把访问量信息保存在应用程序对象中，那么可以使用下面的代码来实现：

```
int count = (int)Application["Count"];
count += 1;
Application["Count"] = count;
```

这段代码看似没有问题，但是其实它设计的并不完美。

如果有多个用户同时请求 Web 服务器，服务器就打开了多个线程，同时处理用户的请求。当有多个用户(比如两个)同时执行到这段代码时，可能同时先执行了第一行代码，取出了应用程序对象中保存的统计信息的值；然后又同时执行了累加的代码；最后可能又先后执行了保存信息的操作。这时理论上 Application["Count"]的值应该增加 2，但是实际执行以后，Application["Count"]的值却只增加了 1。

为了解决这种访问冲突问题，Application 对象提供了一种“加锁”的机制来同步对应用程序状态的访问。这个机制通过 Lock()方法和 UnLock()方法来实现。用户可以把访问和修改应用程序的代码放在 Lock()方法和 UnLock()方法调用之间。这样当一个用户访问应用程序对象 Application 时，首先使用 Lock()方法将 Application 对象锁定，此时其他用户对 Application 对象的访问就必需处于等待状态，直到该用户调用 UnLock()方法释放对 Application 的锁定，其余的用户才能进行访问。

```
Application.Lock();
int count = (int)Application["Count"];
count += 1;
Application["Count"] = count;
Application.UnLock();
```

上述代码解决了应用程序的多线程访问的问题。但是在使用这种方法时，需要注意在 Lock()方法和 UnLock()方法之间应尽可能不要做太多工作。也就是说每一次对 Application 对象执行“加锁”操作的时间段应尽可能短，以提高应用程序的性能。

另外，还要注意一点，即使用完后必须对 Application 对象进行解锁操作，否则其他请求就会一直被阻塞，造成程序假死。最好的方法是使用 try-catch-finally 语句来管理该操作，代码如下：

```
try
{
    Application.Lock();
    int count = (int)Application["Count"];
    count += 1;
    Application["Count"] = count;
```

```
}
catch
{
    //异常处理
}
finally
{
    Application.UnLock();
}
```

这样就能保证无论中间出现什么异常都能够及时地释放对 Application 对象的锁定。

7.3.2 实例描述

Application 对象可以实现在应用程序级的数据保存，通常可以使用它来统计并记录当前系统的在线用户数量。

在本实例中，我们就使用 Application 配合 Web 服务来实现在线用户数量的功能，并在上一节所讲解的计算器窗体中显示出来。

7.3.3 实例应用

【例 7-2】 使用 Application 统计系统的在线人数

(1) 在 Web 服务项目中添加一个“全局应用程序类”(名为 Global.asax)。

(2) 在 Global.asax 文件中修改 Application_Start()方法、Session_Start()方法和 Session_End()方法，分别用于初始化 Application 状态、创建会话时累加在线人数信息、会话到期时递减在线人数信息，代码如下：

```
protected void Application_Start(object sender, EventArgs e)
{
    /* 如果“在线人数”为空，将其初始化为 0 */
    if (Application["CountOnline"] == null)
    {
        Application["CountOnline"] = 0;
    }
}
protected void Session_Start(object sender, EventArgs e)
{
    int countOnline = (int)Application["CountOnline"];      //获得在线人数
    countOnline++;                                           //执行累加
    Application["CountOnline"] = countOnline;                //设置当前在线人数
}
protected void Session_End(object sender, EventArgs e)
{
    int countOnline = (int)Application["CountOnline"];      //获得在线人数
    countOnline--;                                           //执行递减
    Application["CountOnline"] = countOnline;                //设置当前在线人数
}
```

这样就在 Web 服务器端实现了统计活动 Session 个数(即在线人数)的功能。

(3)　这里需要在 Web 服务类中添加 Web 服务方法，用于查询在线人数，代码如下：

```
[WebMethod(EnableSession = true)]
public int GetCountOnline()
{
    return  (int)Application["CountOnline"];
}
```

(4)　需要重新编译服务器端的 Web 服务项目，并在客户端刷新该 Web 引用。

(5)　在客户端窗体中放入一个显示统计信息的 Label 标签 lblCount，以及一个刷新在线人数的按钮 btnRefresh，并添加该按钮的单击事件处理程序，代码如下：

```
private void btnRefresh_Click(object sender, EventArgs e)
{
    int count = wsc.GetCountOnline();
    this.lblCount.Text = count.ToString();
}
```

这样就可以在单击“刷新”按钮的时候在窗体中显示当前在线人数。

7.3.4　运行结果

运行 Web 服务项目。然后再运行客户端窗体，这时程序会自动请求 Web 服务，并创建一个 Session 对象。然后关闭该窗体，重新运行它，程序会再次请求 Web 服务，创建一个 Session 对象，在窗体中单击“刷新”按钮，窗体中将显示统计个数为 2，如图 7-7 所示。



图 7-7　在线用户个数统计

7.3.5　实例分析



源码解析：

本实例，首先在 Web 服务项目中添加一个“全局应用程序类”，然后在该类中的 Application_Start()方法里初始化统计在线人数的应用程序状态。在 Session_Start()方法和 Session_End()方法中分别执行在线人数的更新操作，在 Web 服务中添加查询在线人数的 Web 方法。最后在客户端窗体中刷新对该 Web 服务的引用，并调用该 Web 方法获取在线人数。

7.4　在 Web 服务客户端保存用户状态

Cookie 提供了一种在 Web 应用程序中存储特定的用户信息的方法。我们可以将一些用户状态信息保存到 Cookie 中，然后在客户端每次请求的时候从 Cookie 中取出来进行相应的处理。

下面来介绍 ASP.NET 中的 Cookie 对象。

 视频教学：光盘/videos/07/在客户端保存用户状态.avi　　长度：17 分钟

7.4.1　基础知识——Cookie 对象

Cookie 提供了一种机制，可以在客户端保存用户的状态信息，也可以很方便地实现用户状态信息的管理。

1．什么是 Cookie

Cookie 是一段随着用户请求在 Web 服务器和浏览器之间传递信息的文本。用户每次访问站点时 Web 应用程序都可以读取 Cookie 里的信息。

如果用户访问某个 Web 站点，则服务器返回给用户的不仅是一个用户请求的 Web 页面，还有一个包含日期时间的 Cookie。用户在得到 Cookie 以后，将其存储在本地硬盘中的临时文件夹中。以后该用户再次访问这个 Web 站点时，这组 Cookie 信息会随着用户请求被发送到服务器中。服务器接收到请求并进行处理，然后将处理后的 Web 页面和 Cookie 信息一起返回给用户。

客户端保存的一组 Cookie 信息对应一个 Web 站点。因此，在访问某个网站的任何页面的时候，都可以读取到同一组 Cookie 信息。

当用户访问不同的站点时，各站点都会向用户返回一组 Cookie 信息，用户会分别存储这些 Cookie 信息。

Cookie 可以帮助 Web 站点存储有关访问者的信息。一般来说 Cookie 可以作为一个客户端的名片，为 Web 服务器提供标识信息。因此 Cookie 可以作为一种保持 Web 应用程序执行状态的方法。

2．使用 Cookie 的限制

因为 Cookie 支持在客户端存储用户状态信息，所以客户端在考虑了种种局限性的因素以后，对 Cookie 的使用做出了如下一些限制。

1)　Cookie 大小的限制

多数浏览器支持的 Cookie 最大为 4096 字节(4KB)。由于这种限制，最好使用 Cookie 存储少量的数据。比如用户 ID 或其他身份标识符等。我们可以在服务器端使用 Cookie 中的 ID(或其他身份标识符)来确定用户，并从其他数据源中读取用户信息。

2)　Cookie 个数的限制

浏览器一般来说还对 Cookie 的个数进行了一些限制。多数浏览器只允许每个站点存储 20

个 Cookie，如果存储更多的 Cookie，一些最老的 Cookie 信息便会被自动丢弃。有些浏览器还会对它们管理的 Cookie 总数做出限制，通常来说都是 300 个，所以也不能在个数上过多地使用 Cookie。

3)　浏览器禁用 Cookie

在客户端，用户的浏览器可能设置为禁止使用 Cookie。如果用户设置的安全级别过高，可以通过一些手段来让用户接收你的 Cookie。而如果用户直接拒绝接收 Cookie，我们只能放弃使用 Cookie，而通过其他一些机制来存储用户信息。存储会话状态可以使用会话，但是会话也通常依赖于 Cookie(这一点前面我们也介绍过了)。

虽然 Cookie 在开发应用程序中非常有用，但有些应用程序不应该完全依赖 Cookie 来实现会话状态的存储。

总之，在使用 Cookie 时，需要根据不同情况做出不同的设计。

3．使用 Cookie

客户端的 Web 访问模块(比如浏览器内核)负责管理用户系统中的 Cookie。Cookie 通过服务器将客户端的响应发送到客户端，客户端在执行 Web 请求时将 Cookie 同时发送到服务器中。

在 ASP.NET 中，服务器对客户端的响应使用的是 HttpResponse 对象，可以将 Cookie 信息保存到 HttpResponse 对象中。服务器接收客户端请求使用的是 HttpRequest 对象，可以使用 HttpRequest 对象获取客户端发送过来的 Cookie 信息。

每个 Cookie 都必须有一个唯一的名称，以便以后读取 Cookie 时可以根据名称来识别它。因为 Cookie 是按名称存储，所以相同名称的多个 Cookie 信息只会存储其中的一个。

在 Web 服务中，所有自定义的 Web 服务都继承自 System.Web.Services.WebService 类，而该类中并没有直接使用 Cookie 的内置对象，也没有与之相关的 Request 对象和 Response 对象。但是，System.Web.Services.WebService 类中有一个名为 Context 的对象，可以使用它来获取当前 Web 服务的上下文对象。然后使用 Context 对象来获取 Request 对象和 Response 对象，进而操作 Cookie 集合。示例代码如下：

```
//写入 Cookie
HttpResponse response = Context.Response;                    //得到 Response 对象
HttpCookieCollection writeCookies = response.Cookies;        //得到 Cookie 集合
HttpCookie wCookie = new HttpCookie("username");             //创建 Cookie
wCookie.Value = "admin";                                     //设置 Cookie 的值
writeCookies.Add(wCookie);                                   //添加 Cookie 到集合中
//读取 Cookie
HttpRequest request = Context.Request;                       //得到 Request 对象
HttpCookieCollection readCookies = request.Cookies;          //得到 Cookie 集合
HttpCookie rCookie = readCookies["username"];                //获得 Cookie
string username = string.Empty;
if (null != rCookie)                                     //验证取出的 Cookie 是否为空
{
   username = rCookie.Value;                             //获取 Cookie 的值
}
```



因为 Request 中的 Cookie 对象中的指定 Cookie 可能为空，所以在使用时要进行非空验证，以免出错。

上面使用的 Cookie 是临时 Cookie，它随浏览器的关闭而丢失。不过我们还可以设置 Cookie 的到期时间，为 Cookie 指定生命周期，以便 Cookie 能够永久保存。

这一功能通常使用于页面中用户的配置信息，或者永久保存用户的登录状态。比如要设置一个用户登录一次以后永不过期，可以将 Cookie 的到期时间设置得非常长，例如 20 年或 50 年或者更长。



一般来说，还没有哪个 Web 系统可以持续运行数十年而不更新或者升级，所以这样就达到了一个逻辑上的相对永久保存的状态。另外，用户可以随时清除其计算机上的 Cookie。即便存储的 Cookie 距到期日还有很长时间，但用户还是可以决定删除硬盘中的 Cookie，清除 Cookie 中存储的所有信息。

我们可以通过设置 Cookie 对象的 Expires 属性来为 Cookie 对象指定过期时间，示例代码如下：

```
HttpResponse response = Context.Response;                  //得到 Response 对象
HttpCookieCollection writeCookies = response.Cookies;  //得到 Cookie 集合
HttpCookie wCookie = new HttpCookie("username");          //创建 Cookie
wCookie.Value = "admin";    //设置 Cookie 的值
wCookie.Expires = new DateTime(2012, 12, 21);
                           //设置 Cookie 到期时间为 2012 年 12 月 21 日
writeCookies.Add(wCookie);  //添加 Cookie 到集合中
```

7.4.2 实例描述

如果要将用户状态保存在 Web 服务器端，就可以使用 ASP.NET 提供的 Session 对象。如果需要在服务器端节省内存资源，就可以使用 Cookie 将用户状态信息保存到客户端。这样不仅可以节省用户状态信息，还可以进行更加持久的保存会话状态。因为 Cookie 可以被持久化到客户端的硬盘中。

本实例，我们就使用 ASP.NET 中的 Cookie 对象来持久化保存一个用户的登录状态。

7.4.3 实例应用

【例 7-3】 在 Web 服务客户端保存用户状态

(1) 创建一个提供发送短信服务的 Web 服务，名为 WSMessage。

(2) 在 WSMessage 中，添加两个 Web 方法，分别用于登录和发送短信的功能，代码如下：

```
public class WSMessage : System.Web.Services.WebService
{
   [WebMethod]
   public bool Login(string name,string pwd)
   {
      HttpResponse response = Context.Response;   //获得响应对象
```

```
        HttpCookieCollection wCookies = response.Cookies; //获得 Cookie 集合
        HttpCookie ckLoginUser = new HttpCookie("LoginUser", name); //创建 Cookie
        wCookies.Add(ckLoginUser);  //添加 Cookie
        return true;
    }
    [WebMethod]
    public string SendMessage(string target, string message)
    {
        HttpRequest request = Context.Request;
        HttpCookieCollection rCookies = request.Cookies;
        HttpCookie ckLoginUser = rCookies.Get("LoginUser");
        if (null == ckLoginUser)
        {
            return "用户验证失败，请重新登录。";
        }
        string loginUser = ckLoginUser.Value;
        //发送短信代码(略)
        return "发送成功！";
    }
}
```

在上述代码中，用户登录后并没有使用 Session 来保存用户的登录状态。而是使用 Cookie 将用户状态保存到客户端。这样就可以避免用户长时间没有执行操作而会话丢失的问题。

(3) 在窗体应用程序中添加一个关于该 Web 服务的 Web 引用，名为 Message Service。

(4) 在我们的发送短信的应用程序窗体中声明一个关于 WSMessage 的代理类，并初始化它，代码如下：

```
public partial class SendMessage : Form
{
    MessageSerivce.WSMessage wsm = new MessageSerivce.WSMessage();
    public SendMessage()
    {
        wsm.CookieContainer = new System.Net.CookieContainer();
        InitializeComponent();
    }
    //其他代码略
}
```

(5) 创建一个登录窗体，其中包含一个发送短信的代理类的类变量，并在构造方法中接收一个对象，初始化它，代码如下：

```
public partial class Login : Form
{
    MessageSerivce.WSMessage wsm = null;
    public Login(MessageSerivce.WSMessage wsm)
    {
        this.wsm = wsm;
        InitializeComponent();
```

```
    }
    //其他代码略
}
```

(6) 在登录窗体的登录按钮事件中使用 Web 服务代理类对象执行远程登录功能。如果登录成功，提示用户并关闭该窗体，代码如下：

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string loginName = this.txtLoginName.Text;
    string password = this.txtPassword.Text;
    if (wsm.Login(loginName,password))
    {
        MessageBox.Show("登录成功！");
        this.Close();
    }
    else
    {
        MessageBox.Show("登录失败！");
    }
}
```

(7) 在发送短信的应用程序窗体中执行登录命令时，使用对话框的模式弹出登录窗体，代码如下：

```
private void tsmiLogin_Click(object sender, EventArgs e)
{
    Login loginForm = new Login(wsm);
    loginForm.ShowDialog();
}
```

(8) 在发送短信功能的代码中调用发送短信功能的 Web 服务，代码如下：

```
private void btnSend_Click(object sender, EventArgs e)
{
    string target = this.txtMobile.Text;
    string message = this.txtMessage.Text;
    string result = wsm.SendMessage(target, message);
    MessageBox.Show(result);
}
```

7.4.4 运行结果

首先运行 Web 服务项目。然后运行客户端窗体应用程序，打开 SendMessage 对话框。在该对话框中输入一个手机号码和一段短信内容，单击【发送】按钮，弹出“用户验证失败，请重新登录”的提示框，如图 7-8 所示。

单击【用户登录】菜单项，在弹出的【登录】对话框中输入用户名和密码(这里随意输入就行)，单击【登录】按钮，系统就会调用 Web 服务执行登录操作。登录成功以后程序会弹出提示框，如图 7-9 所示。单击【确定】按钮以后会关闭【登录】对话框。

接下来在 SendMessage 对话框中单击【发送】按钮，系统就会调用远端 Web 服务，并弹出执行结果，如图 7-10 所示。

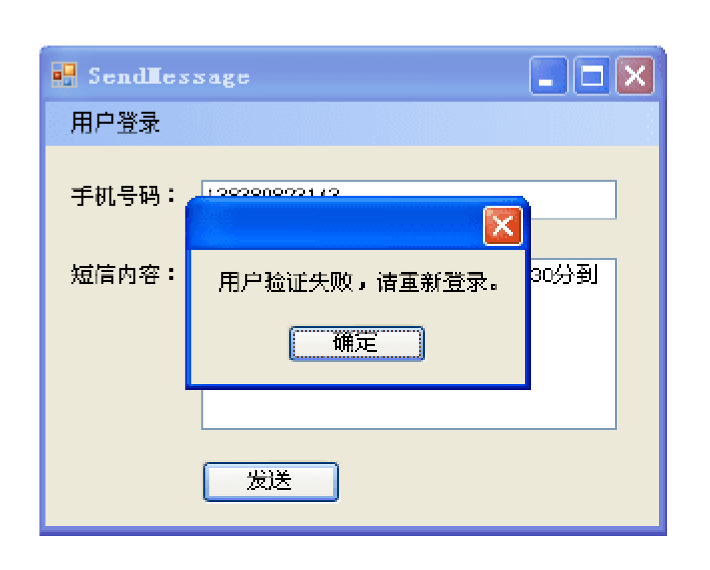



图 7-8　验证失败



图 7-9　登录成功



图 7-10　发送成功

7.4.5　实例分析


源码解析：

本实例，首先在 Web 服务项目创建一个提供发送短信功能的 Web 服务。在该 Web 服务中创建两个 Web 方法，分别用于执行登录功能和发送短信的功能，在用户执行登录操作以后，服务器将用户的状态使用 Cookie 对象保存到客户端。然后在客户端窗体应用程序中添加对该 Web 服务的 Web 引用，并分别调用服务器端创建的 Web 服务来实现登录和发送短信的功能。

7.5　常见问题解答

7.5.1　在 Web 服务中是否可以保持客户端状态



在 Web 服务中可以保持客户端状态吗？
网络课堂：http://bbs.itzcn.com/thread-15249-1-1.html

创建一个类，它的代码如下所示：

```
namespace WinFormsApp
{
    class Class1
    {
        public string name;

        public Class1(string  n)
        {
```

```
            name=n;
        }
    }
    public class Service1 : System.Web.Services.WebService
    {
        public Class1 c;
        public Service1()
        {
            //InitializeComponent();
        }
        [WebMethod]
        public void Sett()
        {
            this.c=new Class1( "test ");
        }
        [WebMethod]
        public string Gett()
        {
            return c.name;
        }
    }
}
```

客户端在调用 Sett()后再调用 Gett()的时候，c 就是 null，如果需要实现保持客户端的状态功能，应该怎么做？

【解决办法】

Web 服务的私有成员对于客户端是没有用的，即在两次调用过程中，服务器端会分别创建 service 对象。所以要使两次调用能访问同一个对象，就需要使用 Session。首先 Web Method 需要改动，然后客户端用 Cookie 来保持两次操作在同一个 Session 中。

例如：

```
//webservice
[WebMethod(true)]
public void Sett()
{
     Class1 c=new Class1( "test ");
     Session[ "MyTest "] = c;
}
[WebMethod(true)]
public string Gett()
{
     Class1 c = Session[ "MyTest "] as Class1;
     if(c != null)
          return c.name;
     else
          return null;
}
```

7.5.2　WebService.asmx 如何使用 Session



WebService.asmx 如何使用 Session 啊？
网络课堂：http://bbs.itzcn.com/thread-15250-1-1.html

为什么在 Web 服务里面使用 Session 记录值后，在第二次访问时就自动消失了呢？

【解决办法】

因为客户端没有接收到服务器回传的 Cookie 值。因此，需要使用一个 Cookie 容器接收服务器的 Cookie，也就是设置代理类对象的 CookieContainer 属性。

Web Server 服务端代码如下：

```
[WebMethod(EnableSession = true)]
public void WriteSession(string text)
{
    Session["a"] = text;
}
[WebMethod(EnableSession = true)]
public string ReadSession()
{
    return Session["a"].ToString();
}
```

客户端代码如下：

```
System.Net.CookieContainer cc = new System.Net.CookieContainer();
localhost.WebService ws = new localhost.WebService();
ws.CookieContainer = cc;
ws.WriteSession(TextBox1.Text);
TextBox2.Text = ws.ReadSession();
```

7.6　习　　题

一、填空题

(1) 在 ASP.NET 应用程序中，所有 Web 服务的父类 System.Web.Services.WebService 中的__________属性可以获得当前会话状态。

(2) 在 Web 应用程序配置文件中，system.web 节点下的__________子节点可以用来设置当前站点的会话状态信息。

(3) 在 ASP.NET Web 应用程序中，__________对象可以在 Web 服务器中共享一些信息，供所有用户请求使用。

(4) 在 ASP.NET Web 应用程序中，HttpCookie 类对象的__________属性可以设置这个 Cookie 的生存周期。

二、选择题

(1) 会话状态可以有 4 个存储方式，其中__________可以将会话存储到另一台服务器中的进程中。

A. Inproc　　B. StateServer　　C. SqlServer　　D. Off

(2) 在 Web 应用程序配置文件中，关于会话的配置节点中，__________属性可以配置会话的生存周期。

A. Mode　　B. Cookieless　　C. Timeout　　D. stateNetworkTimout

(3) 如果要在每一个会话对象创建的时候初始化会话的状态信息，可以在 Global.asax 文件中的__________方法中添加初始化代码。

A. Application_Start　　B. Application_End

C. Session_Start　　D. Session_End

三、上机练习

上机练习 1： 查看当前请求的会话 ID。

本练习要求编写一个 Web 服务，在该 Web 服务中提供一个 Web 方法，用于得到当前浏览器的会话 ID。

需要注意的一点是，本实例要求读者从 Cookie 中取出当前用户的会话 ID，并返回给方法调用者。

运行以后，在浏览器中的效果如图 7-11 和图 7-12 所示。



图 7-11　Web 服务调用方法

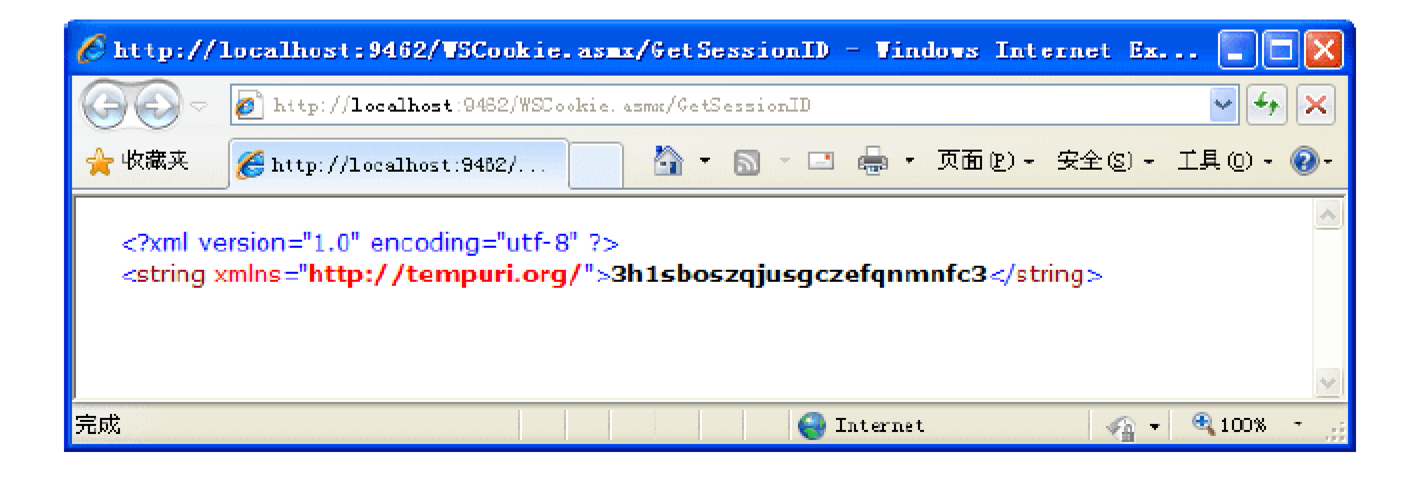


图 7-12　返回当前用户的 SessionID



第 8 章　异 步 服 务

内容摘要：

在使用 Web 应用程序时，通常会因为网络的问题而需要用户进行漫长的等待。同样，Web 服务也不例外。在前面使用 Web 服务的时候其实就已经深有体会。

继续？还是等待？这在设计应用程序的时候是个问题。

实现异步化处理，是开发人员在客户端选择的一个非常好的解决等待问题的方案。对 Web 应用使用异步化请求的方案是可行的：在普通的 Web 应用程序中，使用 Ajax 技术解决了很多请求过程中没必要的等待。.NET 中的 Web 服务也提供了异步调用的机制，可以很轻松地实现 Web 服务的异步化调用。

当使用同步化调用的时候，调用线程会被阻塞，直到调用结束。在一些程序逻辑中，同步化是必需的。比如对数据库的操作，可能需要较长的时间，但是程序逻辑必须在获得数据以后才能继续执行。

异步化调用允许多个线程同时执行。执行调用以后，执行异步化调用的线程不会被阻塞。因此，在执行异步化调用以后，几乎还可以同时执行其他任何操作。

Web 服务在客户端应用的时候支持同步化调用和异步化调用两种使用方式，所以可以根据应用程序的实际需求来决定使用同步方式还是异步方式调用 Web 服务。

在本章，我们就来深入学习基于 ASP.NET Web 服务的同步化调用和异步化调用的主题。在研究 Web 服务的异步化调用时，我们也同时来研究一下.NET Framework 是怎样为其提供支持的。

学习目标：

- 掌握异步化调用 Web 服务的用法
- 掌握异步化调用执行的原理
- 了解异步调用 Web 服务的回调方法
- 了解使用 Web 服务要考虑的一些问题

8.1 Web 服务的性能测试

在使用网络应用程序进行数据同步时，可能会因为网络问题而造成一些延迟。可能我们会因为对网络方面的知识不了解，而对网络延迟这个概念没有很深的印象。

在本节，我们就来对 Web 服务的性能进行一个简单的测试。


视频教学：光盘/videos/08/Web 服务的性能测试.avi　　长度：7 分钟

8.1.1 实例描述

前面我们已经练习过很多 Web 服务的例子，也对 Web 服务的使用方法掌握得相当牢固了。下面以本书第 1 章中使用过的一个执行加法计算的 Web 服务为例，进行一个简单的测试。

8.1.2 实例应用

【例 8-1】 Web 服务的性能测试

(1) 创建一个解决方案，引入第 1 章创建的两个项目：一个执行 Web 服务的 Web 项目，一个使用 Web 服务的控制台项目。

(2) 为了便于观察结果，在 Web 服务的服务器端增加一个执行乘法的 Web 服务方法。修改后，该 Web 服务中的代码如下所示：

```
public class Counter : System.Web.Services.WebService
{
    [WebMethod]
    public decimal Add(decimal n1, decimal n2)
    {
        return n1 + n2;
    }
    [WebMethod]
    public decimal Multiply(decimal n1, decimal n2)
    {
        return n1 * n2;
    }
}
```

(3) 保存以后，需要将该项目重新生成一下。

(4) 在调用 Web 服务的项目中更新 Web 服务。具体操作是在该 Web 服务名称(ServiceCounter)上单击右键，在弹出的快捷菜单中选择【更新服务引用】命令。这样就可以在客户端程序使用该 Web 服务了。

(5) 修改 Program 类中的代码，添加新的程序逻辑，并在调用 Web 服务之前和调用 Web 服务之后输出当前时间信息。代码如下：

```
class Program
```

```
{
    static void Main(string[] args)
    {
        ServiceCounter.CounterSoapClient counter =
            new ServiceCounter.CounterSoapClient();
        Console.Write("请输入一个数：");
        decimal value1 = decimal.Parse(Console.ReadLine());
        Console.Write("请输入第二个数：");
        decimal value2 = decimal.Parse(Console.ReadLine());
        Console.WriteLine(DateTime.Now);
        var result1 = counter.Add(value1, value2);
        var result2 = counter.Multiply(value1, value2);
        Console.WriteLine(DateTime.Now);
        var result = result1 == result2;
        Console.WriteLine("这两个数的和和这两个数的积 " + (result ? "相等" : "不相
等"));
        Console.ReadLine(); //程序暂停，等待用户回车确认
    }
}
```

代码输入完毕后，执行存盘操作即可执行该测试了。

8.1.3　运行结果

首先要运行 Web 服务的服务器端项目，然后将解决方案中的控制台项目设置为启动项目，并运行该项目。

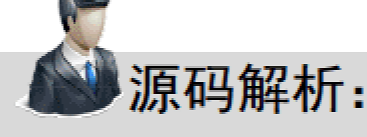
控制台项目运行以后，分别输入两个十进制数，系统会两次调用 Web 服务，并输出结果以及程序运行的当前时间，如图 8-1 所示。



图 8-1　Web 服务的性能测试

从图 8-1 中可以看到，Web 服务是相当耗费时间的。在这个实例中，只用 Web 服务执行了两次简单的算术运算，就足足使用了 3 秒钟的时间。

8.1.4　实例分析



源码解析：

本实例使用第 1 章中的一个实例。首先为其添加了一个执行乘法运算的 Web 服务，并生成项目使其生效；然后再调用 Web 服务的客户端刷新对该 Web 服务的引用，并重新根据程序

逻辑修改调用 Web 服务的代码；在代码中，分别在调用 Web 服务时和调用之后，输出系统的当前时间，以便判断 Web 服务的执行时间。

据运行结果分析，使用 Web 服务确实会造成相当大的时间延迟。

8.2　实现异步调用 Web 服务验证用户注册信息

通过上一节中的例子，我们知道在使用 Web 服务时可能会因为网络的原因造成不同程度的延迟。如果在这时选择等待，可能会使应用程序造成很大的性能浪费，所以建议在一些必要的时候使用异步 Web 服务来提升应用程序的性能。

在.NET 中，使用 Web 服务非常简单。同样，使用异步服务也非常简单。本节中，我们就来详细了解一下在.NET 中如何开发使用异步 Web 服务的应用程序。



视频教学：光盘/videos/08/异步调用 Web 服务.avi　　　　长度：13 分钟

8.2.1　基础知识——异步调用 Web 服务

一般情况下，当程序执行了调用 Web 服务的语句后，客户端会一直等待 Web 服务执行完成并返回结果，然后才继续执行。这样就会造成客户端的线程一直阻塞，从而给用户造成线程假死的现象。这时使用异步调用的方式使用 Web 服务就显得十分有用了。

在使用异步调用的方式使用 Web 服务时，不会让调用 Web 服务的客户端处于阻塞状态，从而避免了线程假死的情况，因此可以在调用 Web 服务的同时做一些其他的处理。



如果要使用 Web 服务，不需要对服务器端的 Web 服务特别设计。因为 Web 服务通信基于 SOAP 消息交换，所以其在本质上就是异步的。修改代理层，就可以同步或异步调用任何 Web 服务。

在.NET 中，使用自动生成的代理类包括了异步调用的任何方法的基本特性。不过在使用异步 Web 服务添加服务引用时需要进行一些简单的配置。具体操作如下。

首先在客户端程序中添加服务引用，打开【添加服务引用】对话框，如图 8-2 所示。

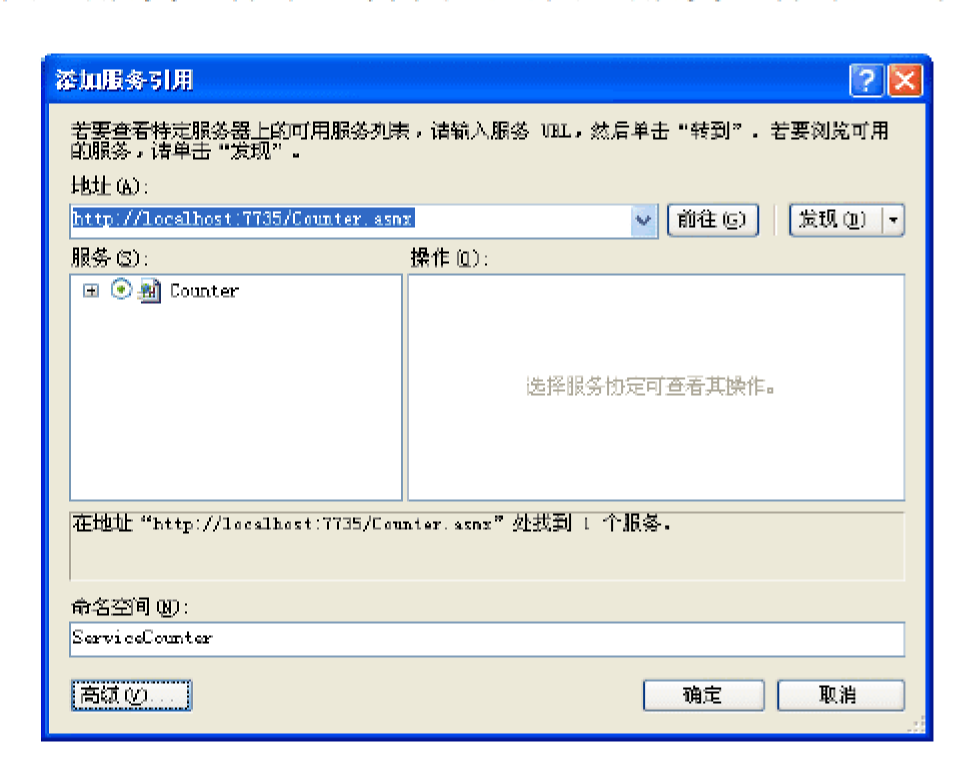


图 8-2　添加服务引用

在【添加服务引用】对话框中单击左下角的【高级】按钮，打开【服务引用设置】对话框。在【服务引用设置】对话框中选中【生成异步操作】复选框，如图 8-3 所示。

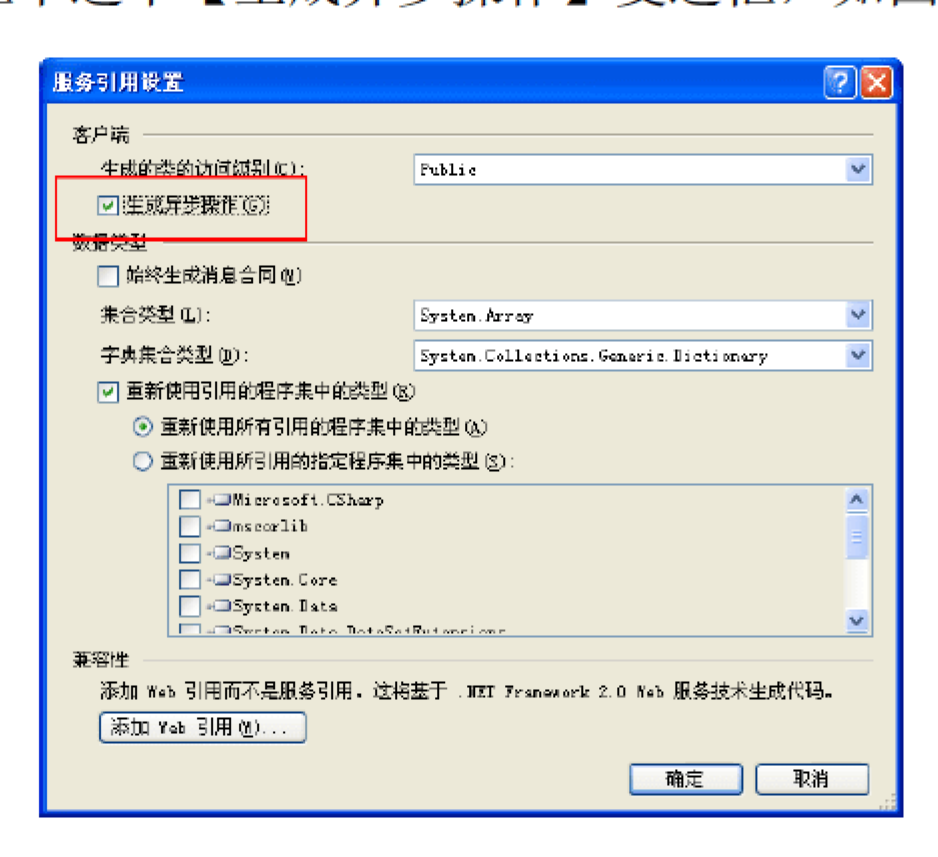


图 8-3　添加引用设置

单击【确定】按钮关闭该对话框。然后在【添加服务引用】对话框中单击【确定】按钮即可创建包含异步请求配置的 Web 服务引用。

在生成的代理类中，相对于没有使用异步 Web 服务创建的代理类而言，使用异步 Web 服务创建的代理类要多出几个方法。下面我们通过 Visual Studio 中的对象浏览器来查看这个类中的接口。图 8-4 所示为没有使用异步方式创建的代理类的方法结构。图 8-5 所示为使用异步方式创建的代理类的方法结构。

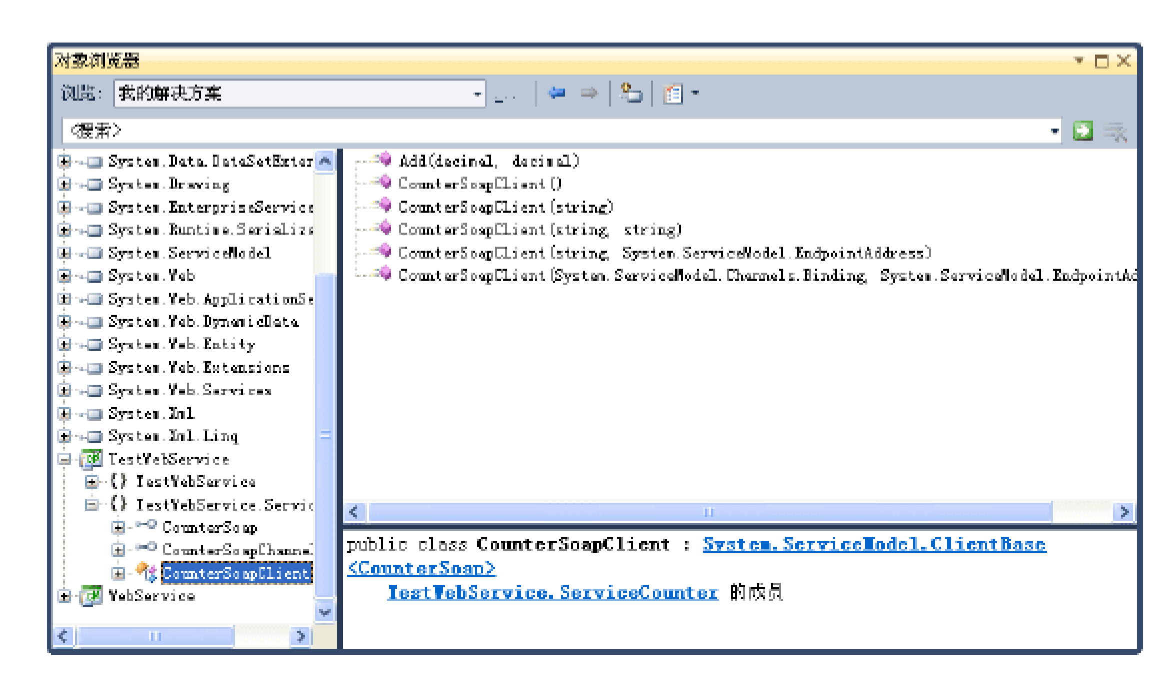


图 8-4　未使用异步方式创建的代理类

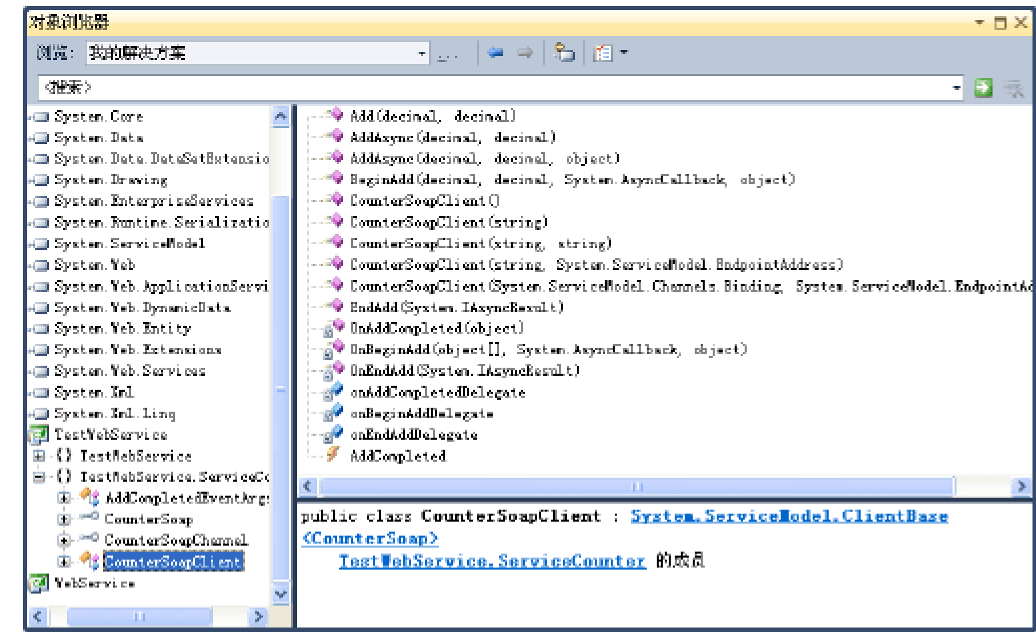


图 8-5　使用异步方式创建的代理类

当然，不同的方式生成的代理类的代码并不相同。以常用的几个方法为例，没有使用异步方式创建的代理类代码如下：

```
public decimal Add(decimal n1, decimal n2) {
    return base.Channel.Add(n1, n2);
}
```

而使用异步方法生成的代理类的主要代码多出来两个方法，如下所示：

```
public decimal Add(decimal n1, decimal n2) {
    return base.Channel.Add(n1, n2);
}
```

```
[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Advanced)]
        public System.IAsyncResult BeginAdd(decimal n1, decimal n2,
System.AsyncCallback callback, object asyncState) {
            return base.Channel.BeginAdd(n1, n2, callback, asyncState);
        }


[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Advanced)]
        public decimal EndAdd(System.IAsyncResult result) {
            return base.Channel.EndAdd(result);
        }
```

上面代码中的第一个方法和未使用异步方式生成的代理类的方法相同。我们可以使用这个方法调用 Web 服务并等待返回结果。后面两个方法是为进行异步交互提供的，使用它们可以轻松地实现对 Web 服务的异步调用。



这里的 Begin***()方法通常提供两个额外的参数。以 Web 服务中的 Add 方法为例，除了 Add 方法需要的两个 decimal 类的方法以外，它还需要提供两个用于同步的参数。但是与之对应的 End***()方法通常只需要一个参数。

这里仍然以前面使用过的加法计算器为例。前面已经添加了允许异步请求的 Web 服务引用。下面直接修改客户端类 Program 中的代码，使用异步方式请求执行加法的 Web 服务，如下所示：

```
class Program
{
    static void Main(string[] args)
    {
        ServiceCounter.CounterSoapClient counter =
            new ServiceCounter.CounterSoapClient();
        Console.Write("请输入加数：");
        decimal value1 = decimal.Parse(Console.ReadLine());
        Console.Write("请输入被加数：");
        decimal value2 = decimal.Parse(Console.ReadLine());
        var result = counter.BeginAdd(value1, value2, null, null);
                                                //执行异步请求
        Console.WriteLine("本地运行的相加结果：" + (value1 + value2));
        Console.WriteLine("等待 Web 服务请求结果");
        while (!result.IsCompleted)
        {
            Console.Write(".");
            System.Threading.Thread.Sleep(10);
        }
        Console.WriteLine();                    //换行
        var value3 = counter.EndAdd(result);
        Console.WriteLine("Web 服务请求结果：" + value3);
        Console.ReadLine();                     //程序暂停，等待用户回车确认
    }
}
```



上面代码中为了防止延迟过长、输出过多的英文句号，使用 System.Threading.Thread.Sleep(10); 语句将当前线程每次都延迟 10 毫秒。

上面代码执行以后，结果如图 8-6 所示。

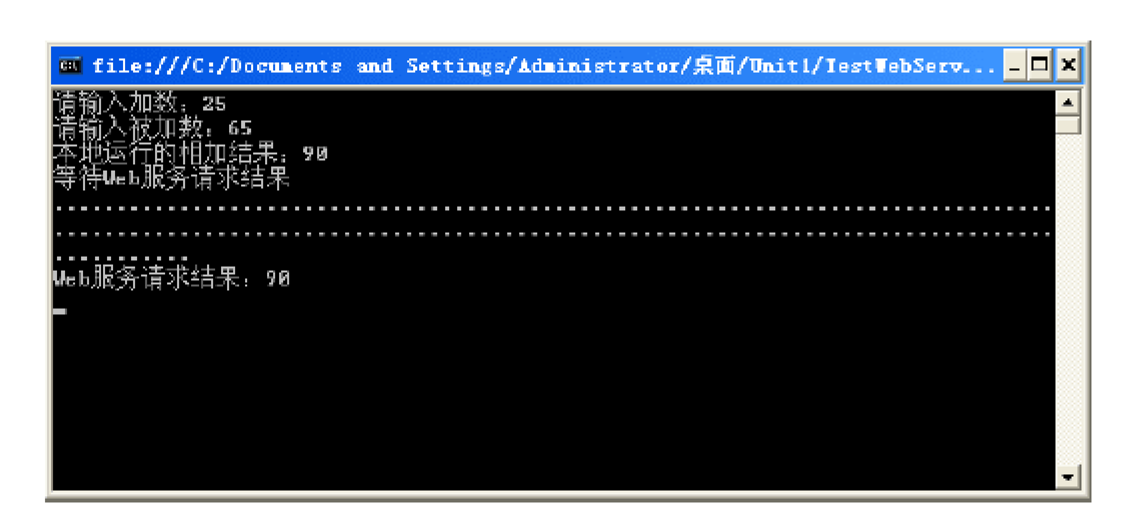


图 8-6　异步调用 Web 服务的执行过程

8.2.2　实例描述

微软发布了一个 Web 服务的典型应用——Passport。其实现思路是将所有用户信息作为一个信息中心保存在一个提供 Web 服务的服务器中，用户可以远程使用这些 Web 服务提供的接口来进行用户的信息和权限的验证。

本实例中，使用 Web 服务模拟一个用户信息中心。在另一个 Web 系统中的用户注册功能实现过程中使用该 Web 服务完成用户的注册。

8.2.3　实例应用

【例 8-2】 实现异步调用 Web 服务验证用户注册信息

(1)　在 Web 服务的 Web 项目中创建一个执行用户管理功能的 Web 服务 Passport。

(2)　在 Web 服务 Passport 中分别创建一个验证用户是否存在的 Web 服务方法，和一个执行注册的方法，Passport 类的代码如下：

```
public class Passport : System.Web.Services.WebService
{
    [WebMethod]
    public bool ValidateUsername(string username)
    {
        return "admin" == username;
    }
    [WebMethod]
    public void Register(string username, string password, string realName)
    {
        //执行注册的代码(略)
    }
}
```

这里的两个 Web 服务方法的实现只是模拟了一下相应的功能，没有具体的实现。

(3) 在客户端的 Web 项目中添加对该 Web 服务的引用。

(4) 在用户注册页面添加页面结构，主要代码如下所示：

```
<table>
<tr><td>用户名：</td><td>
    <asp:TextBox ID="txtUsername" runat="server"
CssClass="input"></asp:TextBox></td></tr>
<tr><td>登录密码：</td><td>
    <asp:TextBox ID="txtPassword" runat="server" CssClass="input"
TextMode="Password"></asp:TextBox></td></tr>
<tr><td>真实姓名：</td><td>
    <asp:TextBox ID="txtRealname" runat="server"
CssClass="input"></asp:TextBox></td></tr>
<tr><td colspan="2">
    <asp:Button ID="btnSave" runat="server" Text="注册"
onclick="btnSave_Click" /></td></tr>
<tr><td colspan="2">
    <asp:Label ID="lblError" runat="server"
ForeColor="Red"></asp:Label></td></tr>
</table>
```

(5) 在页面中【注册】按钮的单击事件中获取页面中用户输入的内容，并执行相应的验证过程和注册执行过程，代码如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    string username = this.txtUsername.Text.Trim();
    string password = this.txtPassword.Text.Trim();
    string realname = this.txtRealname.Text.Trim();
    ServicePassport.PassportSoapClient passport =
        new ServicePassport.PassportSoapClient();
    IAsyncResult validateUsername = null;
    if (string.Empty != username)
    {
        //开始验证
        validateUsername = passport.BeginValidateUsername(username, null,
null);
    }
    else
    {
        this.lblError.Text = "用户名不能为空";
        return;
    }
    if (string.Empty == password)
    {
        this.lblError.Text = "密码不能为空";
        return;
    }
    if (string.Empty == realname)
    {
```

```
        this.lblError.Text = "真实姓名不能为空";
        return;
    }
    while (!validateUsername.IsCompleted)
    {
        System.Threading.Thread.Sleep(10);
    }
    //结束验证，获取验证结果
    var result = passport.EndValidateUsername(validateUsername);
    if (result)
    {
        this.lblError.Text = "用户" + username + "已存在";
        return;
    }
    else
    {
        //使用 Web 服务注册用户信息
        passport.Register(username, password, realname);
        this.lblError.Text = "注册成功";
    }
}
```

在这段代码中，因为考虑到在验证用户名是否存在时可能需要较长的时间。所以，这里使用异步调用 Web 服务的方式验证用户是否存在，并同时对用户输入的其他项目进行验证。

(6)　保存项目，即可完成测试。

8.2.4　运行结果

首先运行 Web 服务项目，然后再运行 Web 客户端项目，并访问用户注册页面，这里访问 http://localhost:10587/Register.aspx。

在用户注册页面，输入用户名、密码和真实姓名，然后单击【注册】按钮。程序执行完信息验证，在页面中输出相应的错误信息，如图 8-7 所示。



图 8-7　用户名验证失败

换一个用户名(比如“joker”)，再次单击【注册】按钮，页面中将提示“注册成功”信息，如图 8-8 所示。



图 8-8　注册成功

8.2.5　实例分析



源码解析：

本实例，首先在 Web 服务项目中创建一个用于执行用户管理功能的 Web 服务 Passport，并为 Passport 添加两个 Web 服务方法，分别执行用户名验证和用户注册功能；然后在 Web 项目中添加对该 Web 服务的引用，并在页面中使用异步方法验证用户信息，同时执行对其他输入项目的验证；最后在验证通过时使用 Web 服务提供的注册方法执行用户注册功能。

8.3　异步调用和同步调用的比较

虽然异步调用和同步调用在表面上十分相似，但是用来调用 Web 服务的内在过程是不同的。

在使用 Web 服务的系统中，客户机调用的服务器的方法被捆绑在 SOAP 消息中，并通过 Internet 使用 HTTP 协议发送到服务器端。

由于 Internet 固有的延迟特性，使用远程过程调用(RPC)在访问 Web 服务时会因为不同的网络环境而产生不同的效果。所以，在设计一些使用 Web 服务的应用时，不得不考虑一些调用方法选择的问题。

下面就针对客户端调用 Web 服务的这两种方法进行比较，来理解异步调用 Web 服务的深层运行机制。

视频教学：光盘/videos/08/异步和同步的比较.avi　　**长度：**10 分钟

异步调用 Web 服务的方式在处理比较复杂的业务逻辑时，往往会节省大量时间，比同步调用 Web 服务更有效率。这一点，在前面两节的实例中已经有所体会。

不过同步也有同步的好处，比如在处理一些获得结果才能继续执行的业务逻辑时，同步的方式使用更加方便。

下面就根据它们的执行原理进行一些比较。

1．同步化调用 Web 服务

同步化操作由一组前后紧接的组件或方法的调用组成。一个同步化操作会阻塞整个调用 Web 服务的进程，直至 Web 服务调用结束，下一行操作才会执行。

这种行为模式有很多实例。比如我们对数据库执行查询操作，可能这个查询操作十分复杂，也或许数据库服务器受网络影响，响应速度较慢。但是在数据库服务器返回结果之前，我们必须处于等待状态。等数据库查询到相应的数据，并做出响应以后，调用 Web 服务的客户端进程才会解除阻塞状态。同步调用 Web 服务的执行原理如图 8-9 所示。

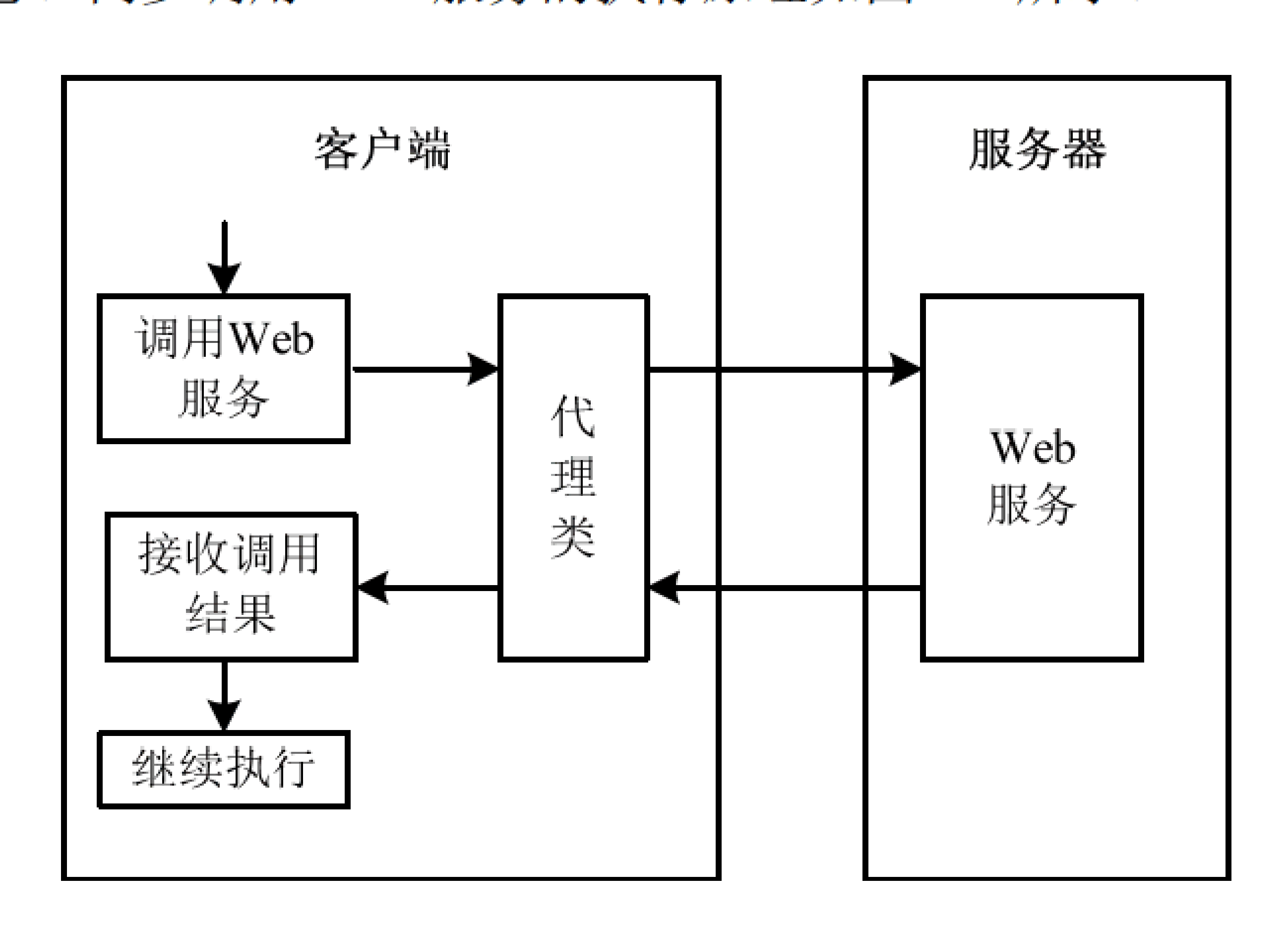


图 8-9　同步调用 Web 服务的执行原理

从图 8-9 中可以看到，同步调用的过程只有一条主线。从调用，到服务器执行并返回结果，是一气呵成，没有分支。

(1) 程序执行到调用 Web 服务的代码以后，操作代理类；

(2) 代理类请求 Web 服务器，调用 Web 服务，并使该线程进入等待状态；

(3) Web 服务在接收到客户端的请求以后，执行相应的操作，并返回执行结果；

(4) 代理类在接收到 Web 服务的响应信息以后，解析并将解析到的数据返回给方法调用者；

(5) 主程序接收到调用结果以后，继续执行后面的代码。

在大多数情况下，同步调用所产生的性能损耗是可以接受的，但是这样没有办法保证在处理并发操作时的额外系统开销。

2．异步化调用 Web 服务

异步化调用是客户端的行为，类似于 Web 开发中的 Ajax 技术。

异步化调用的一大特点就是异步化操作不会阻塞启动调用操作的线程，不影响调用线程的

其他业务逻辑的执行。

不过异步化调用的应用程序必须通过反复检测代理类请求 Web 服务的执行状态来确定 Web 服务是否执行完成。如果执行完成，才可以从代理类实例中获取异步调用的执行结果。异步调用 Web 服务的执行原理如图 8-10 所示。

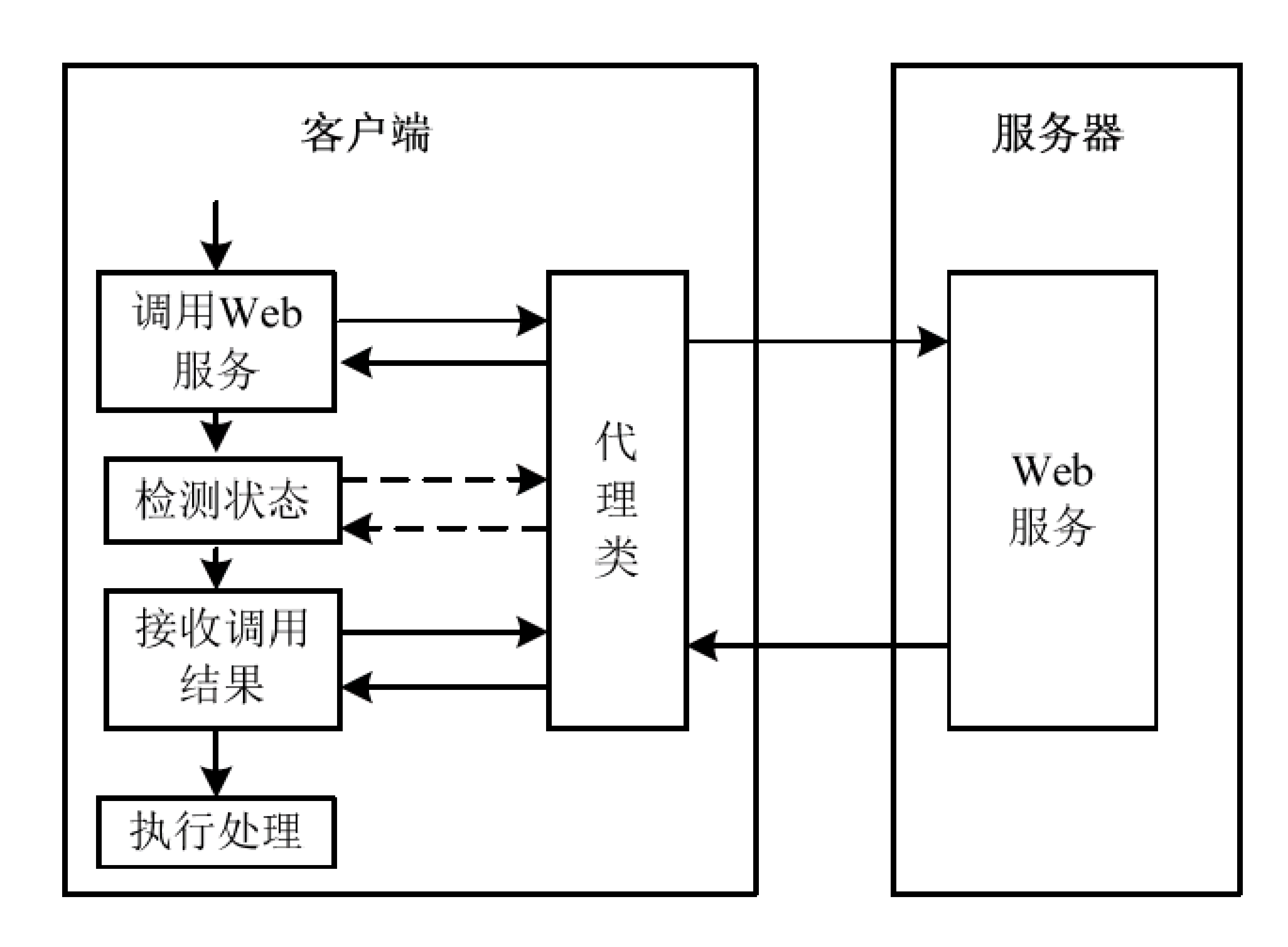


图 8-10　异步操作的执行原理

从图 8-10 可以看到，异步操作也非常简单。

(1) 应用程序在代理类中返回给调用者一个操作句柄，并同时调用 Web 服务；

(2) 应用程序在获得操作句柄以后，继续执行其他业务逻辑；

(3) 应用程序在执行的时候，可以不定时地反复检测代理类异步请求的状态；

(4) 在检测到异步请求 Web 服务的操作完成后，应用程序线程就可以操作代理类，获取相应的执行结果；

(5) 最后应用程序处理异步 Web 服务调用得到的数据。

异步调用操作相对于同步调用来说，操作稍微有一点复杂。不过，在处理一些复杂的业务逻辑时，合理地使用异步调用 Web 服务的方式为应用程序带来了性能上的提升。所以在设计一个应用程序时，使用同步调用还是异步调用 Web 服务需要考虑应用程序的实际需求。

3．何时使用异步化调用 Web 服务

其实在.NET 中，几乎所有的方法调用都是使用同步处理的方式执行的，这样可以很轻松地避免异步方式调用产生的数据同步、共享资源的访问等问题。但是，使用异步的方式确实可以在很大程度上解决程序性能的问题，这一点是同步方式所不具备的优势。

不过在.NET 中，使用最先进的技术，也可以成功地开发出一些很完美的使用异步化的解决方案。下面来列举几个需要使用异步方式的情况。

- 在开发 Windows 应用程序时。如果在 Windows 应用程序的主线程中使用同步方式调用 Web 服务，那么在数秒之内，该 Windows 应用程序的主窗口将处于阻塞状态，给用户造成应用程序“假死”的感觉。这个时候，使用异步调用的方式可以很轻松地避免这个问题。

- 如果某个 Web 服务的调用需要一定的时间来完成，那么使用异步化调用的方式将非常有益。这样使用异步化调用方式以后，在 Web 服务响应请求之前，还可以做一些其他操作，以节省时间，提高应用程序的性能。
- 客户端程序需要同时调用多个 Web 服务时，使用异步化调用的方式再好不过了。这样多个 Web 服务的调用操作同时执行，可能仅仅使用一个调用的时间，就可以完成所有的调用。
- 如果 Web 服务的客户端是一个 ASP.NET 应用程序，使用异步操作会是一个不错的方案。因为同步化调用会导致线程阻塞，线程阻塞以后 ASP.NET 工作站线程将会停止运行。那么，在该工作站下运行的其他 Web 应用程序也将被同时阻塞。

8.4　异步用户信息查询服务

前面我们通过一些例子学习了.NET 下的 Web 服务的异步调用的方法。下面，我们将深入研究.NET Framework 中的公共语言运行时(Common Language Runtime，CLR)是如何为异步化调用 Web 服务提供支持的。

 **视频教学：**光盘/videos/08/异步用户信息查询服务.avi　　长度：13 分钟

8.4.1　基础知识——使用回调

对于程序员来说，编写一个同步化的方法调用是非常简单的事情，但是编写异步化的方法调用却有些复杂。特别是在.NET Framework 出现以前，编写异步化，对程序员来说简直就是噩梦。

还好，.NET Framework 对解决上述问题提供了基本支持。我们可以在.NET 中轻松地实现对 Web 服务的异步调用。

1. .NET CLR 提供的机制

.NET Framework 提供了功能强大的多线程机制和异步化的机制，可以在.NET 中很轻松地实现异步化操作。

异步化编程是.NET Framework 中的诸多领域都会用到的一种特性，包括网络通信、信息队列、异步委托、IO 操作等。

.NET Framework 致力于为异步化处理提供一个一致的框架，其提供了一个通用的设计模式作为核心理念。该设计模式包含的几个基本概念如下。

- .NET Framework 为异步化编程提供必要的服务。
- 在使用异步化时，决定是否使用异步化方式的选择权由客户端来决定。
- 客户端可以自由选择是否支持异步化调用。
- 客户端没有必要为使用异步化而编写额外的代码。
- CLR 为异步化提供了类型安全。

因为.NET 中编写的程序都要运行在 CLR 中，所以我们在.NET 中不必为使用异步化而改

为使用特定的语言。.NET 中使用的任意一门语言，都可以使用 CLR 提供的支持来编写异步化功能。

2．认识回调

前面我们演示过，异步化编程的必要条件之一就是必须使调用线程知道异步请求何时结束。当然，可以像前面使用过的方法一样反复检测异步请求的状态来实现这个功能。与这种方法相比，另一种方法就是使用一个带有回调方法的接收器对象，它允许调用异步请求的对象通知客户机应用程序异步调用执行结束。这种使用回调方法的机制在.NET Framework 中的具体实现就是委托(delegate)。

委托是对函数的封装，是一种引用方法的类型。一旦为该委托分配方法，那么该委托将与该方法具有相同的行为。

创建用来接收回调的方法必须使用 System.AsyncCallback 委托定义的签名。以本章第 2 节中的实例，执行用户注册验证的异步回调功能来说，可以定义如下回调方法：

```
public void Callback(IAsyncResult result)
{
}
```



由于 ASP.NET WebForm 程序有页面生存周期的问题，因此在下面的实例中将使用 WinForm 程序来演示。

如果要附加回调，则需要创建一个指向该回调方法的 AsyncCallback 委托变量，然后将这个变量传递给 BeginValidateUsername()方法。代码如下：

```
…
if (string.Empty != username)
{
    AsyncCallback ac = new AsyncCallback(Callback);
    passport.BeginValidateUsername(username, ac, null); //开始验证
}
else
{
    this.lblError.Text = "用户名不能为空";
    return;
}
…
```

这个回调方法只能接收结果，但是它无法知道这个响应来自哪个请求。所以，为了解决这个问题，.NET 允许在 IAsyncResult.AsyncState 属性中发送随结果对象提供的额外信息。该对象是一个 Object 类型，所以它能够存放任意类型的实体。我们可以在使用 BeginValidateUsername 方法时为其最后一个参数指定 IAsyncResult.AsyncState 属性的值。然后在回调方法中获取这个值，进行相应的类型转换即可使用，修改后的代码如下：

```
…
if (string.Empty != username)
{
```

```
    AsyncCallback ac = new AsyncCallback(Callback);
    passport.BeginValidateUsername(username, ac, passport); //开始验证
}
else
{
    this.lblError.Text = "用户名不能为空";
    return;
}
…
```

然后可以在回调方法中执行相应的业务逻辑，代码如下：

```
public void Callback(IAsyncResult handler)
{
    ServicePassport.PassportSoapClient passport =
        (ServicePassport.PassportSoapClient)handler.AsyncState;

    //结束验证，获取验证结果
    var result = passport.EndValidateUsername(handler);
    if (result)
    {
        this.lblError.Text = "用户" + this.txtUsername.Text.Trim() + "已存在";
        return;
    }
    else
    {
        //使用 Web 服务注册用户信息
        passport.Register(this.txtUsername.Text.Trim()
            , this.txtPassword.Text.Trim()
            , this.txtRealname.Text.Trim()
            );
        this.lblError.Text = "注册成功";
    }
}
```

这个例子可以像前面所列举的那个注册信息的 Web 页面一样，请求并异步访问 Web 服务。

不过，设计的这个例子有一个潜在的疏漏。因为回调方法运行的线程和调用代码的线程不同，所以在设计应用程序时还要考虑线程同步的问题。关于线程同步的问题，并不是三两句话就能解释清楚的，所以这里我们不再研究。要想了解更多的方法，可以参考专门讲解线程的书。

8.4.2　实例描述

在 ASP.NET 应用程序中使用回调的方式处理 Web 服务的异步调用是很不明智的选择。因为往往 Web 服务调用返回结果之前，ASP.NET 的页面请求也过了生存周期。在 WinForm 应用程序中却不存在这个问题。

在一个客户端的 WinForm 应用程序中使用 Web 服务的方式与服务器通信，是一个非常不错的选择。

实例将在一个查询用户信息的客户端应用程序中使用异步回调的方式来查询用户信息。

8.4.3 实例应用

【例 8-3】 异步用户信息查询服务

(1) 创建一个查询用户信息的 Web 服务，名为 Users。

(2) 在 Users 中添加一个执行查询操作的 Web 服务方法 Find。Find 方法接收一个用户 ID 作为参数，并返回指定用户的信息。代码如下：

```
public class Users : System.Web.Services.WebService
{
    [WebMethod]
    public Models.Users Find(int id)
    {
        if (2 == id)
        {
            return new Models.Users()
            {
                ID = 2,
                Name = "张月",
                Age = 23,
                Email = "zhangyue@zy.com",
                Phone = "13333333333",
                Sex = "男",
                Remark = "清华大学计算机 2302 届毕业生"
            };
        }
        return null;
    }
}
```

这里我们没有使用数据库，只是在 Find 方法中模拟查询数据。如果调用该方法的参数为 2，则返回一个用户信息对象，否则返回 null。



在该方法中使用的 Models.Users 类只是一个封装数据的模型类，其中的代码并不复杂，所以这里不再展示该模型类的详细代码。

(3) 在客户端的一个 Windows 窗体应用程序中，添加对该 Web 服务的异步引用，命名空间名称为 ServiceUsers。

(4) 在客户端，有一个执行用户查询功能的窗体，现在为其【查询】按钮的单击事件添加异步查询用户信息的代码，如下所示：

```
private void btnQuery_Click(object sender, EventArgs e)
{
    int id = 0;
    try { id = int.Parse(this.txtID.Text); }
    catch
    {
        MessageBox.Show("请输入一个合法的用户 ID! ");
        return;
    }
    ServiceUsers.UsersSoapClient usc = new ServiceUsers.UsersSoapClient();
    AsyncCallback ac = new AsyncCallback(FindUserCallback);
    usc.BeginFind(id, ac, usc);
}
```

(5) 前面，在创建 AsyncCallback 类对象时用到了一个回调方法 FindUserCallback。该回调方法用于执行异步 Web 服务请求完成以后的操作。代码如下：

```
public void FindUserCallback(IAsyncResult handler)
{
    ServiceUsers.UsersSoapClient usc =
        handler.AsyncState as ServiceUsers.UsersSoapClient;
    ServiceUsers.Users user = usc.EndFind(handler);
    if (null != user)
    {
        this.txtID.Text = user.ID.ToString();
        this.txtName.Text = user.Name;
        this.txtAge.Text = user.Age.ToString();
        this.txtEmail.Text = user.Email;
        this.txtPhone.Text = user.Phone;
        this.txtRemark.Text = user.Remark;
    }
    else
    {
        MessageBox.Show("没有找到该用户! ");
    }
}
```

(6) 当然，这里同样会面临线程间操作共享数据的问题。这里的解决方案是在该窗体的构造方法中设置 CheckForIllegalCrossThreadCalls 属性值为 false，代码如下：

```
public UserForm()
{
    CheckForIllegalCrossThreadCalls = false;
    InitializeComponent();
}
```

这样，在执行多线程操作窗体中的对象时就不再进行线程验证，也就避免了“线程间操作无效”的问题。

注意　虽然设置窗体的 CheckForIllegalCrossThreadCalls 属性会解决在该窗体下的多线程操作的问题，但是为了线程的安全性，一般不建议使用该方法解决多线程互访的问题。

8.4.4　运行结果

首先运行 Web 服务项目，然后运行客户端的窗体程序。在编号文本框中输入一个不存在的用户编号(比如 3)，单击【查询】按钮，将弹出“没有找到该用户”的提示框，如图 8-11 所示。

在【编号】文本框中输入要查询的编号 2，然后单击【查询】按钮，系统将查询出编号为 2 的用户信息，并显示到窗体中，如图 8-12 所示。

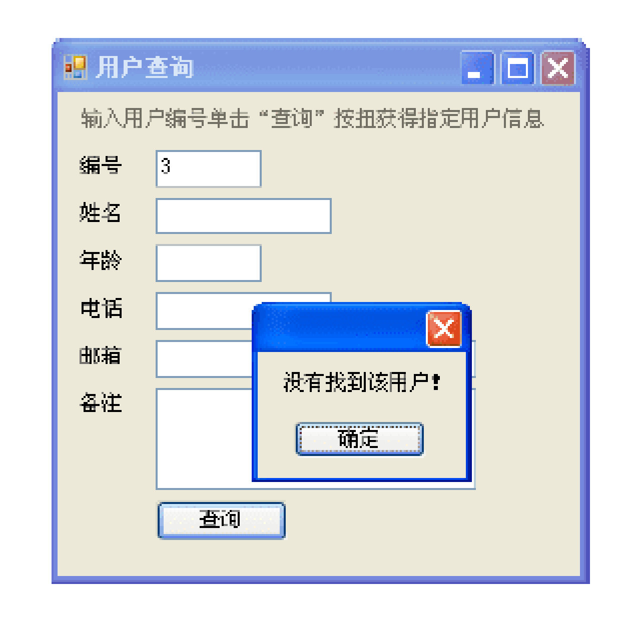


图 8-11　没有找到该用户

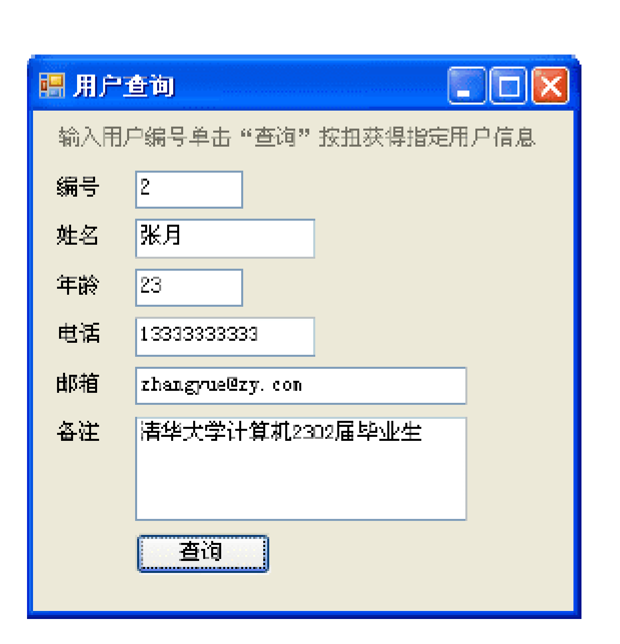


图 8-12　查看用户信息

8.4.5　实例分析



源码解析：

本实例，首先创建一个提供用户信息查询服务的 Web 服务，并模拟一些查询逻辑；然后在客户端添加对该 Web 服务的引用；为客户端中的【查询用户信息】窗体中的【查询】按钮的单击事件添加处理程序；接着编写一个处理异步查询用户信息结果的回调方法；最后需要设置窗体的 CheckForIllegalCrossThreadCalls 属性来解决多线程的共享数据访问问题。

8.5　设计 Web 服务需要考虑的事项

在使用 Web 服务时，因为访问 Web 应用程序的一些不可避免的问题，所以在调用 Web 服务时不得不为一些特殊情况制定一些处理方案。

本节对调用 Web 服务时的超时问题的处理进行研究。



视频教学：光盘/videos/08/设计 Web 服务的问题.avi　　长度：10 分钟

8.5.1　基础知识——超时问题的处理

因为 Web 应用程序由于网络原因有延迟的特性，所以在设计使用 Web 服务的应用程序时需要考虑到因为网络延迟而带来的一些异常问题的处理。

1．超时问题的处理

当需要在客户端采用同步化调用 Web 服务的模式，并且这个 Web 服务是一个在网络中的 Web 服务器时，就必须考虑这个客户端应用程序怎样能够比较健壮地运行。也就是说要考虑到客户机的网速，或者 Web 服务器的运行负荷等问题。

如果遇到响应时间过长的情况，我们不可能允许客户端应用程序一直处于等待状态。在遇到服务器超时的问题时，可以使用超时时间来限定客户端请求 Web 服务等待的最长时间。使用 Visual Studio 生成的 Web 服务的代理类中包含了调用 Web 服务使用的那些方法以外的其他方法和属性，Timeout 属性就是其中之一。

Timeout 属性的格式如下：

```
WebService ws = new WebService();
//设置代理类请求超时时间为 3 秒
ws.Timeout = 3000;
```

Timeout 属性是基类 WebClientProtocol 中声明的属性，经过多层的派生关系，被包含到代理类中。



在.NET Framework 4.0 中，使用一个 Web 服务可以选择“添加 Web 引用”也可以选择“添加服务引用”。这里要注意，在使用“添加服务引用”生成的代理类中没有 Timeout 属性。所以需要选择“添加 Web 引用”来引入一个 Web 服务。

Timeout 属性指定了 Web 服务的客户端在等待同步化请求 Web 服务的最大超时时间。该属性的默认值为-1，表示无限大。

2．关闭请求

在客户端应用程序中使用异步化的方式请求 Web 服务时，如果回调机制在一定的时间内没有返回值，则必须考虑应用程序处理这种超时问题的方法。

我们可以控制在调用该操作时的终止时间，并继续执行其他操作。



因为 Web 架构服务的特殊性，所以没有办法终止服务器端 Web 服务方法的执行。

代理类从 ClientBase 类中继承了两个方法 Abort()和 Close()，可以用来结束该异步服务的执行。

这两个方法的功能差不多，都可以结束当前异步调用 Web 服务的操作。它们在 ClientBase 类中的声明如下：

```
// 摘要：
// 使 System.ServiceModel.ClientBase<TChannel> 对象立即从其当前状态转换到关闭状态。
```

```
    public void Abort();
//
// 摘要:
// 使 System.ServiceModel.ClientBase<TChannel> 对象从其当前状态转换到关闭状态。
public void Close();
```



这里需要注意的是，在使用 Close()方法时可能会执行失败而抛出异常。

8.5.2 实例描述

本节学习了在使用 Web 服务的过程中遇到请求超时的问题的解决方案。我们可以设置同步请求 Web 服务的超时时间，也可以在异步请求 Web 服务时执行放弃请求的操作。

本实例就以网站管理系统的用户登录功能为例，简单演示一下同步请求 Web 服务的超时时间的设置方法。

前面的章节中，我们在实现用户注册功能的时候使用了一个名为 Passport 的 Web 服务，所以这里也使用该服务执行用户登录功能的验证。

8.5.3 实例应用

【例 8-4】 访问网络中的 Web 服务

(1) 在前面创建的 Web 服务 Passport 中创建一个接收登录信息验证的 Web 方法，代码如下：

```
[WebMethod]
public bool Login(string username,string password)
{
    System.Threading.Thread.Sleep(10000);
    return true;
}
```

这里在进行登录验证的时候，让应用程序请求暂停 10 秒钟。

(2) 在客户端应用程序中添加 Web 引用，引用目标为前面创建的 Web 服务 Passport，Web 服务的引用名称为 Services.Passport。

(3) 在客户端应用程序的登录页面 Login.aspx 中，为【登录】按钮添加单击事件处理程序，代码如下：

```
protected void imgbtnLogin Click(object sender, ImageClickEventArgs e)
{
    WebService.Passport.Passport passport = new
WebService.Passport.Passport();
    passport.Timeout = 3000;    //设置延迟等待时间为 3 秒
    bool loginResult = false;
    try
    {
```

```
        loginResult = passport.Login(this.txtLoginName.Text, this.txtPassword.
          Text);
    }
    catch
    {
        ScriptManager.RegisterStartupScript(this, this.GetType(), "", "alert
          ('Web 服务请求超时！')", true);
        return;
    }
    if (loginResult)
    {
        ScriptManager.RegisterStartupScript(this, this.GetType(), "", "alert('
          账户验证成功！')", true);
    }
    else
    {
        ScriptManager.RegisterStartupScript(this, this.GetType(), "", "alert('
          账户验证失败！')", true);
    }
}
```

在创建代理类对象时设置其最大等待的延迟时间，在这里设置 Timeout 属性的值为 3000，即 3 秒钟。如果验证超时，系统将会弹出“请求超时”的提示信息。

8.5.4　运行结果

首先，运行 Web 服务所在的 Web 项目。

然后，访问客户端的登录页面，分别在【用户】和【密码】文本框中输入一些文本(这里只为测试，所以没有进行实际的验证)，然后单击【登录】按钮，执行登录请求。Web 页面在等待 3 秒钟以后，将认为验证登录信息的 Web 服务请求超时，就会弹出“请求超时”的提示框，如图 8-13 所示。



图 8-13　Web 服务请求超时

8.5.5 实例分析



源码解析：

本实例首先要在提供登录验证的 Web 服务 Passport 中添加一个执行登录验证的 Web 方法 Login，并在该方法执行时延迟 10 秒钟；然后在客户端中添加关于该 Web 服务的 Web 引用；在客户端登录页面中的【登录】按钮的单击事件中添加验证的业务逻辑；接着，创建代理类对象，使用 Timeout 属性设置请求延迟为 3 秒钟；最后使用同步方式调用 Web 服务时，该请求将最多等待 3 秒钟就结束请求。

8.6 常见问题解答

8.6.1 .NET 中 Web 服务是同步调用还是异步调用



.NET 中 Web 服务是同步调用还是异步调用？

网络课堂：http://bbs.itzcn.com/thread-15200-1-1.html

.NET 中 Web 服务是同步调用还是异步调用？谁能给我说说它们的区别。

【解决办法】

在.NET 中调用 Web 服务，可以使用同步方式，也可以使用异步方式。同步就是一次执行完了，异步就是多个执行动作是分开的。两种方式可以由用户自由选择，根据实际情况来决定使用那种方式。

8.6.2 关于 Web 服务异步回调的问题



关于 Web 服务异步回调的问题。

网络课堂：http://bbs.itzcn.com/thread-15201-1-1.html

我在 Web 服务中写了一个返回 DataSet 的方法。在 Client 端用异步回调的方法在 DataGrid 中显示该 DataSet 中的数据。数据取出来了，但总是绑定不到 DataGrid 中。如果不是异步，而是同步的话是可以的。同样对于 DataSet 绑定方法，同步能够显示，而异步就不能显示了，有没有人遇到跟我同样的问题啊？

【解决办法】

这里有个区别：主线程和后台线程。主线程和 UI 有关，用来刷新页面和绑定数据。而异步调用是后台线程，它无法刷新页面和数据源。

这里需要做一些特殊的处理。在回调方法中，需要用当前 Form 实例的 Invoke 调用一个委托，在该委托方法中设置 DataGrid 的 DataSource。

8.7 习　　题

一、填空题

(1) 在客户端项目中添加服务引用时，需要在【服务引用设置】对话框中选中__________单选按钮，可以添加支持异步调用的服务引用。

(2) 如果 Web 服务中有一个名为 Insert 的 Web 服务方法，那么在客户端添加异步服务引用以后，可以使用代理类对象的__________方法来启动对该 Web 服务方法的异步调用。

(3) 代理类对象在使用自动生成的方法以异步方式调用一个 Web 服务时，该方法会返回一个__________类的对象。

二、选择题

(1) 使用回调的方式异步调用一个名为 List 的 Web 服务方法时，需要使用__________类封装一个回调方法，并作为参数传递给 BeginList 方法。

A. System.Web.AsyncCallback

B. System.Web.Callback

C. System.AsyncCallback

D. System.Callback

(2) 在使用异步方式调用 Web 服务时，IAsyncResult 类的对象可以使用__________方法获取当前请求的执行状态。

A. Completed

B. IsCompleted

C. Over

D. IsOver

(3) 在使用同步方式请求 Web 服务时，为了防止使用 Web 服务过程中的请求耗时太长而影响应用程序的执行效率，可以设置该代理类对象的_________属性来为其指定一个过期时间。

A. TimeOver

B. OverTime

C. Time

D. Timeout

三、上机练习

上机练习 1： 使用回调方式获取 Web 服务中的用户列表信息。

本章已经对 Web 服务的异步调用的方式进行了比较详细的了解。在 Windows 窗体应用程序中，使用异步回调的方式请求 Web 服务是一种很好的解决方案。这样可使程序逻辑更加清晰合理，也提高了应用程序的执行性能。

本练习中就使用异步回调的方式请求 Web 服务，获取一个用户信息的列表，并显示到窗体中。

为了更加人性化地提示用户等待，在窗体中使用每秒钟刷新的耗时时间来提高应用程序界面的友好性。

图 8-14 是该应用程序的执行结果。

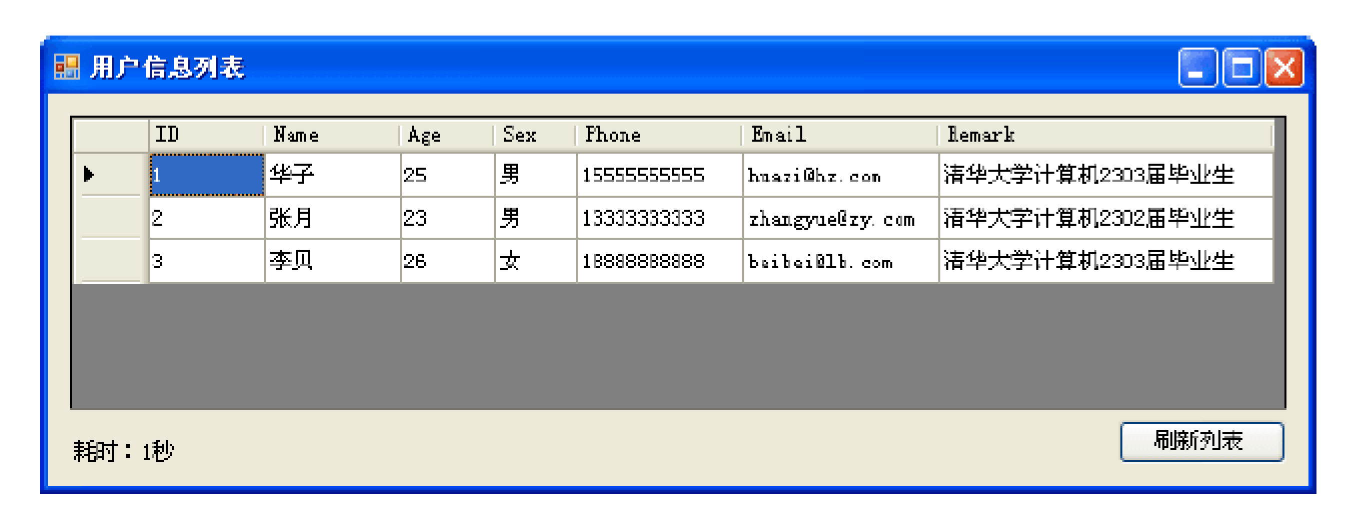


图 8-14　用户信息列表



第 9 章　利用 ASP.NET 的缓存和事务功能

内容摘要：

借助于 ASP.NET 强大的功能，开发人员创建 Web 服务也变得很容易，而且在 Web 服务中可以集成 ASP.NET 的大部分功能。例如，在第 7 章介绍了 Session 和 Application 在 Web 服务中的应用。

本章将讨论如何为 Web 服务添加缓存和事务功能。这两个方面在 Web 开发时最容易被忽略。我们总是在完成了项目的其他方面之后才会想到它们，而不是从一开始就考虑。相反，如果缓存和事务在项目中的应用恰到好处，那么会大大提高代码的性能，避开性能瓶颈，并且使 Web 服务具有可伸缩性。

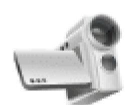
学习目标：

- 了解 ASP.NET 的缓存机制
- 掌握如何为 Web 服务启用输出缓存
- 掌握应用程序缓存的使用方法
- 熟悉 Cache 对象的操作
- 了解如何设置缓存项的过期和依赖性
- 掌握回调函数的使用
- 了解事务的概念及在 Web 服务中的应用

9.1　了解 ASP.NET 缓存机制

所谓缓存，就是指在内存中暂时保存一些经常使用的数据。这样当下次请求这些数据时，就可以直接从内存中获取它们，而不用重复执行产生这些数据的操作，因而可以有效地提高应用程序的性能。

在 ASP.NET 中，开发人员可以通过多种方式来实现缓存功能。例如，可以把一些重复使用的数据保存在 Application 对象中，或者使用专门的缓存对象来保存等。

**视频教学：**光盘/videos/09/了解 ASP.NET 缓存机制.avi　　长度：4 分钟

为了提高应用程序的性能，ASP.NET 使用两种基本的缓存机制来提供缓存功能。第一种是输出缓存，即保存对页面的输出，并在用户再次请求时，重用所保存的输出，而不是再次处理该页面。第二种机制是应用程序缓存，它可以缓存所生成的任何数据，如 DataSet 或自定义报表业务对象。

下面来详细了解一下 ASP.NET 的这两种机制。

1. 输出缓存

输出缓存是指在内存中存储处理后的 ASP.NET 页面内容。这一机制允许 ASP.NET 向客户端发送页面响应，而不必再次经过页面生命周期。

输出缓存对于那些不经常更改，但需要大量处理才能创建的页面特别有用。例如，如果创建需要频繁更新数据的页面时，输出缓存可以极大地提高该页的性能。

如图 9-1 所示为 ASP.NET 输出缓存的执行流程。

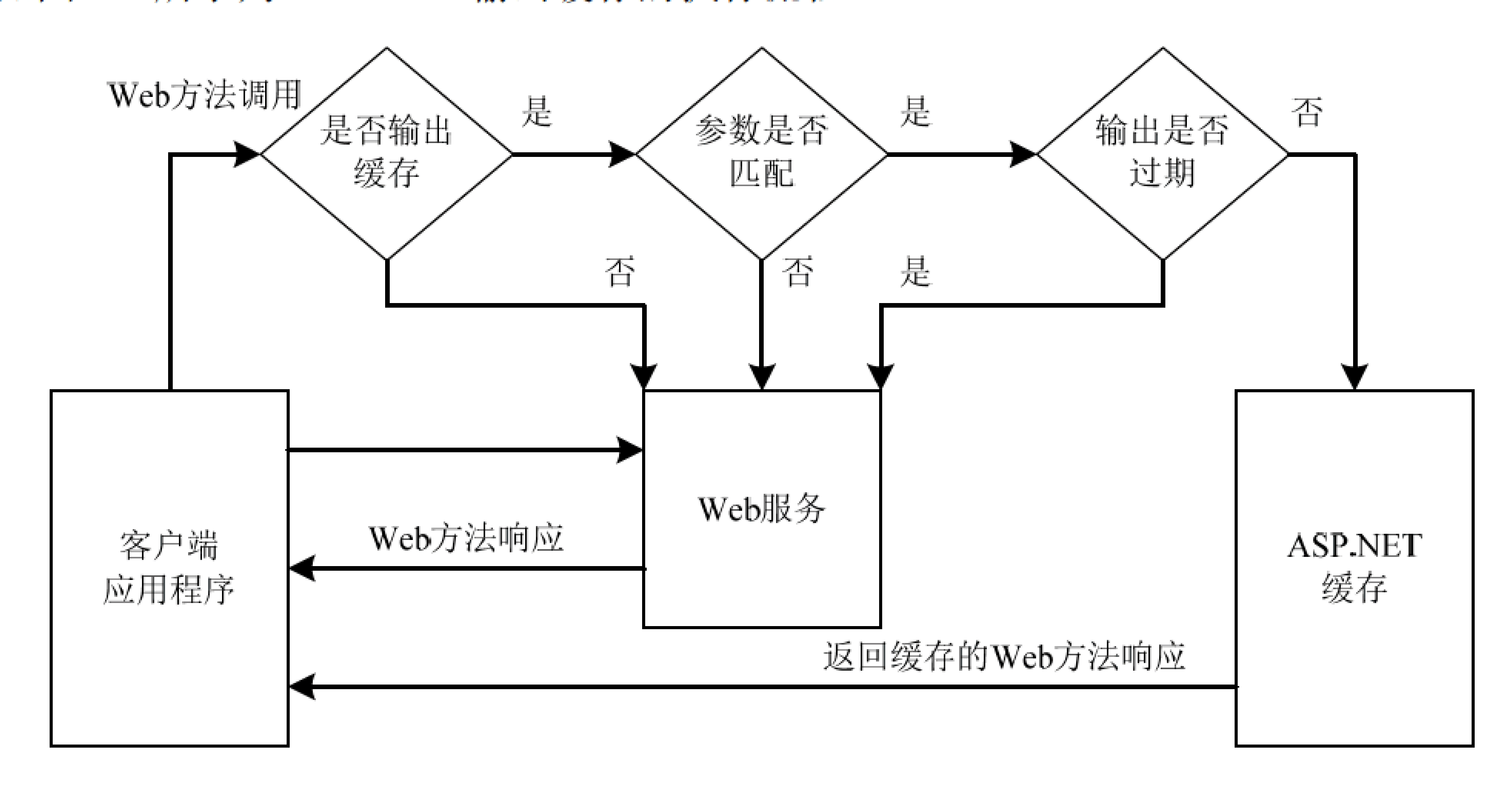


图 9-1　输出缓存执行流程

ASP.NET 输出缓存又提供了两种缓存模型：整页缓存和部分页缓存。整页缓存允许将页面的全部内容保存在内存中，并用于完成客户端请求。部分页缓存允许缓存页面的部分内容，其他部分则为动态内容。

2．应用程序缓存

应用程序缓存为开发人员提供了一种以编程方式控制缓存的机制，它通过“键/值”可以将任意数据存储在内存中。

使用应用程序缓存的流程是，在访问某一项时判断该项是否存在于缓存中，如果存在，则使用。如果该项不存在，则可以重新创建该项，然后将其放回缓存中。这一过程可确保缓存中始终有最新的数据。

如图 9-2 所示为 ASP.NET 应用程序缓存的执行流程。

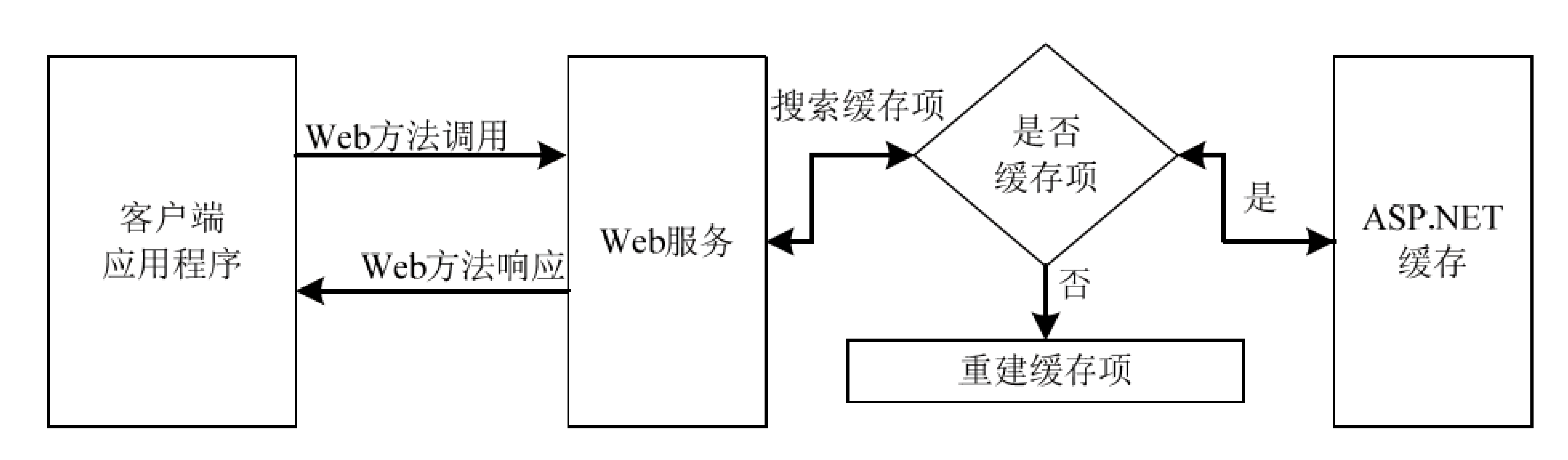


图 9-2　应用程序缓存执行流程

使用应用程序缓存的优点是由 ASP.NET 管理缓存，它会在过期、无效或内存不足时移除缓存中的项。还可以配置应用程序缓存，以便在移除项时通知应用程序。

9.2　使用输出缓存保存文件内容

通过使用输出缓存，可以把为某个特定页面建立的响应暂存起来，当 ASP.NET 接收到该页面的后续请求时，就可以直接返回响应，而不用再去执行 ASP.NET。

在 Web 服务中使用输出缓存的方法非常简单，只需为 Web 服务方法添加 CacheDuration 即可，下面来详细了解一下吧。

 **视频教学：**光盘/videos/09/使用缓存保存文件.avi　　　　长度：11 分钟

9.2.1　基础知识——使用输出缓存

如果要使一个 Web 服务方法可以使用输出缓存，所需的工作仅仅是为 WebMethod 添加 CacheDuration 选项。CacheDuration 的值是一个整数，以秒为单位，用于指定缓存的有效期限。即在这个期限内输出缓存一直有效，超过该期限时将重新执行生成新的输出缓存。

输出缓存的一个最典型示例就是创建一个 Web 服务方法返回当前时间。假设这个方法的代码如下所示：

```
[WebMethod(Description = "测试 ASP.NET 的输出缓存。\n 输出调用时的时间，结果将缓存
 30 秒。",CacheDuration=30)]
public string OutputCache()
{
    return "你访问的时间是：" + DateTime.Now.ToString();
}
```

在第一次请求 OutputCache()方法时，方法内的代码会执行，并返回包含当前时间的字符串。此时，该字符串将在内存中保存 30 秒，30 秒内再次请求 OutputCache()方法时将直接输出该字符串，而不会重新获取时间；当超过 30 秒时，将产生新的输出缓存。



也可以在 Web 服务的 asmx 文件中通过 OutputCache 属性来启用输出缓存。

例如，在一个 ASP.NET 项目中引用 OutputCache()方法所在的 Web 服务，引用名称为“localhost”；然后通过如下的代码来测试 OutputCache()方法：

```
protected void Page_Load(object sender, EventArgs e)
{
    //页面加载完成后调用
    localhost.Service1 ts = new localhost.Service1();
    string strTime1=ts.OutputCache();
    Response.Write("第 1 次调用 OutputCache()方法的结果：<br/>" + strTime1);
    //等待 10 秒后调用
    System.Threading.Thread.Sleep(10000);
    string strTime2 = ts.OutputCache();
    Response.Write("<hr/>10 秒后调用 OutputCache()方法的结果：<br/>" + strTime2);
    //等待 30 秒后调用
    System.Threading.Thread.Sleep(30000);
    string strTime3 = ts.OutputCache();
    Response.Write("<hr/>30 秒后调用 OutputCache()方法的结果：<br/>" + strTime3);
}
```

由于 OutputCache()方法会将结果缓存 30 秒，所以上述代码中 strTime1 和 strTime2 的值是相同的。最后一次调用时，已经距结果产生有 40 秒了，所以此时缓存将过期，重新执行 OutputCache()方法将结果保存到 strTime3 中。

上述代码运行后的结果如图 9-3 所示。

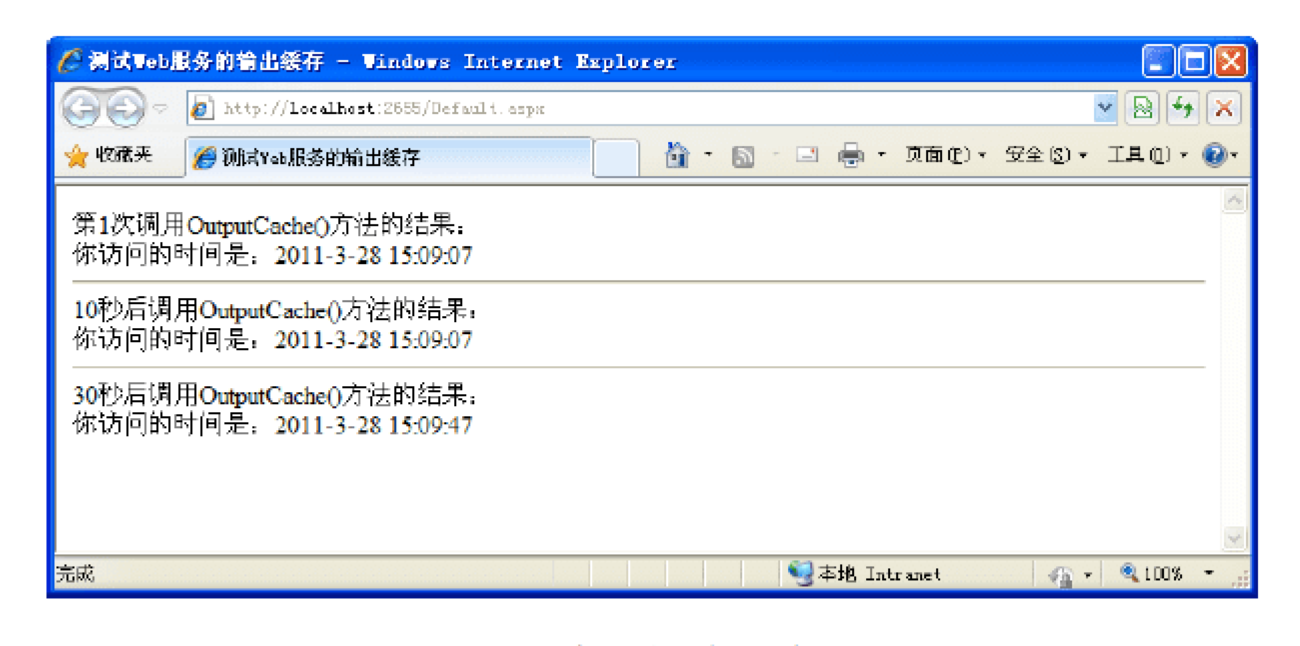


图 9-3 输出缓存运行结果

通过上面简单的示例可以看出，在使用 Web 服务的输出缓存时，我们实际上并不需将任何缓存代码插入 Web 服务方法本身。仅仅是添加一个 CacheDuration 选项，剩下的工作由 ASP.NET 自动完成并管理输出缓存。

9.2.2　实例描述

最近接手了一个项目，需求不复杂，但是对响应速度有比较高的要求。因此，在开发时就针对程序编码、数据库设计和系统架构等方面进行了特别的优化处理。

另外，客户希望具有比较高的跨平台和伸缩性。对于这一点，我们准备使用 ASP.NET Web 服务来做。至于要提高响应速度，没有比缓存更合适的了。

下面就以此项目为例，介绍 Web 服务缓存在项目中的具体应用。希望能对大家有所帮助。

9.2.3　实例应用

【例 9-1】 使用输出缓存保存文件内容

(1) 创建一个 Web 服务项目，并添加一个名为“Service1.asmx”的文件。

(2) 在项目中要对文件进行操作，所以这一步先添加下面的引用。

```
using System.IO;
```

(3) 编写读取文件内容的 Web 服务方法，该方法接受要读取的文件路径，并返回一个自定义的文件实例。实现代码如下：

```
[WebMethod(Description="读取指定路径的文件内容，并缓存 10 分钟。",
CacheDuration=600)]
public Files getFileContent(string filePath)
{
    Files f = new Files();
    if (File.Exists(filePath))                          //判断文件是否存在
    {
        FileInfo info = new FileInfo(filePath);         //获取文件信息
        f.FileName = info.Name;
        f.ExtendName = info.Extension;
        f.LastWriteTime = info.LastWriteTime;
        f.Length = info.Length;
        StreamReader rdFile = new StreamReader(filePath,
System.Text.Encoding.Default);
        string content = rdFile.ReadToEnd();            //获取内容
        rdFile.Close();                                 //关闭文件
        f.FileContent = content;                        //保存内容
    }
    return f;                                           //返回文件实例
}
```

在上述代码中设置 CacheDuration 的值为 600，即表示缓存 10 分钟。filePath 为一个文件的路径，程序先判断该文件是否存在，如果存在则获取文件的内容，并最终返回。

(4) 上一步创建的 getFileContent()方法返回为一个 Files 类的实例。该类是我们创建的自定义类，它的声明及包含的成员如下：

```
public class Files
{
    public string FileName;                                    //文件名称
    public string ExtendName;                                  //文件扩展名
    public DateTime LastWriteTime;                             //上次修改时间
    public string FileContent;                                 //文件内容
    public long Length;                                        //文件大小
}
```

(5) 至此，Web 服务的代码就完成了，是不是很简单啊。下面创建一个 ASP.NET Web 项目对它进行调用。对 ASPX 页面进行简单的修饰，如下所示是与文件显示有关的布局，把它放在页面的合适位置。

```
<div class="contentarea">
    <h1><asp:Literal ID="ltName" runat="server"/></h1>
    <h2>扩展名：<asp:Literal ID="ltExt" runat="server"/> | 文件大小：<asp:Literal
ID="ltLength" runat="server"/>字节 | 上次修改时间：<asp:Literal ID="ltTime"
runat="server"/></h2>
    <p>
    <asp:Literal ID="ltContent" runat="server"/>
    </p>
</div>
```

(6) 添加对 Service1.asmx 的 Web 引用，引用的名称为“localhost”。

(7) 进入页面的 Page_Load()方法，编写代码通过 localhost 调用 Web 服务的 getFileContent()方法。然后根据 getFileContent()方法的返回结果在页面上显示所需的信息，最终代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    localhost.Service1 s = new localhost.Service1();    //实例化 Web 服务类
    localhost.Files f = new localhost.Files();          //实例化 Files 类
    string fileURL = Server.MapPath("Story.txt");       //指定一个文件的完整路径
    f = s.getFileContent(fileURL);              //调用 getFileContent()方法
    ltContent.Text = f.FileContent;             //文件内容
    ltExt.Text = f.ExtendName;                  //文件扩展名
    ltLength.Text = f.Length.ToString();        //文件大小
    ltName.Text = f.FileName;                   //文件名称
    ltTime.Text = f.LastWriteTime.ToString("yyyy-MM-dd");   //文件上次修改时间
}
```

(8) 保存上述对页面所做的修改，完成实例的制作。

9.2.4　运行结果

首先通过浏览器执行 Web 服务文件 Service1.asmx，并调用 getFileContent()方法查看返回的结果，如图 9-4 所示。

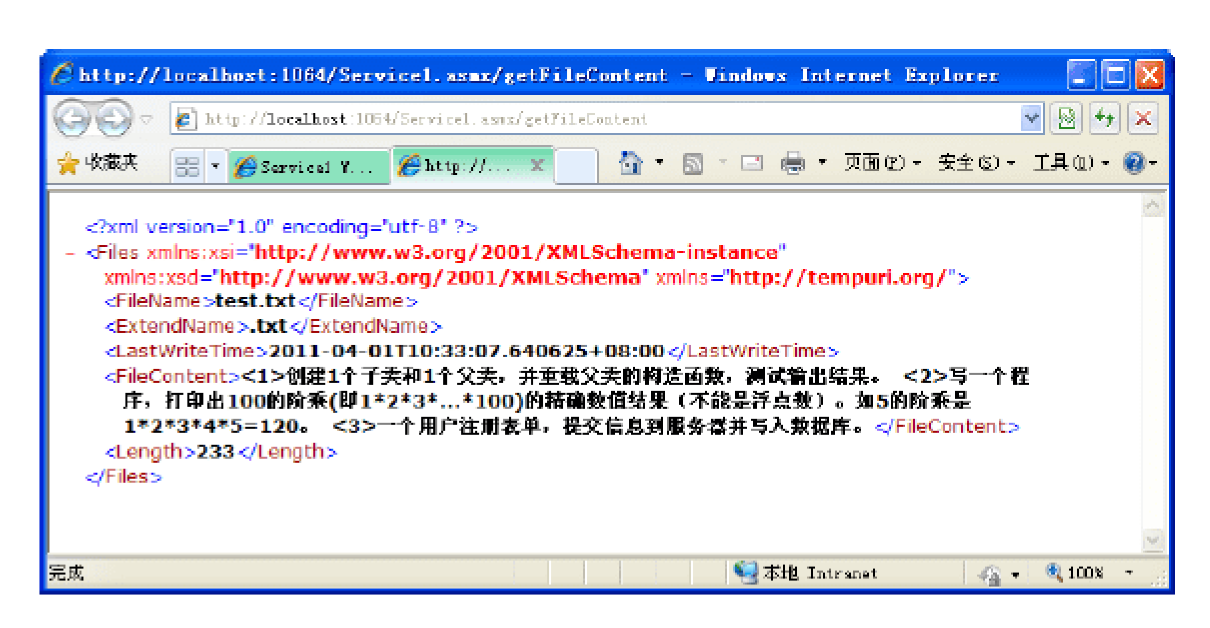


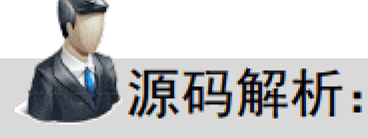
图 9-4　测试 getFileContent()方法

从图 9-4 所示的结果中，可以看到 getFileContent()方法能够返回指定文件的信息。下面通过执行 ASP.NET 页面来测试，运行效果如图 9-5 所示。



图 9-5　查看文件内容

9.2.5　实例分析



源码解析：

本实例我们针对缓存需求，在 Web 服务的方法中通过 CacheDuration 选项设置缓存为 600 秒，即 10 分钟。之后对于缓存便不需要更多的控制，完全由 ASP.NET 来管理。

实例运行后，可对 Story.txt 文件中的内容进行修改。并再次执行 ASP.NET 页面，会发现在 10 分钟之内显示的内容都与图 9-5 所示相同。这也说明了输出缓存起到了存储数据的作用。当超过 10 分钟之后再次刷新页面，此时将显示最新的文件内容并再次缓存 10 分钟。

这种机制可以有效地减少频繁对文件执行读取操作造成的网络堵塞，从而提高响应速度。

9.3 管理缓存的数据

在使用 CacheDuration 选项启用输出缓存时，ASP.NET 将会为每个 Web 服务方法单独开辟缓存空间，也就是说缓存仅针对该方法有效。

如果希望在应用程序的生命周期内共享缓存中的响应，则必须使用应用程序缓存。例如，可以将应用程序内那些频繁访问的数据，以及那些需要大量时间来创建的数据存储在内存中，从而提高性能。


视频教学：光盘/videos/09/管理缓存的数据.avi　　长度：16 分钟

9.3.1 基础知识——使用应用程序缓存

输出缓存只存储 Web 服务方法的结果，而使用应用程序缓存可以存储任何类型的信息。

应用程序缓存实际上是由属于 System.Web.Caching 命名空间的 Cache 类实现的。通过对 Cache 类的应用，可轻松实现添加、检索和移除应用程序数据缓存，以及移除缓存项时通知应用程序等功能。

Cache 类是一个简单字典集合，在 Web 服务中可以通过“this.Context.Cache”来访问这个类的对象实例，并且可以像使用 Application 和 Session 对象一样来访问 Cache 对象。



不要把 Context.Cache 对象与 Context.Response.Cache 对象相混淆。前者是 System.Web.Caching.Cache 类的一个实例，后者则引用了 System.Web.HttpCache Policy 类。而且 HttpCachePolicy 用于为 Web 页面配置输出缓存，对于 Web 服务来说毫无用处。

Cache 对象与 Application 对象相比，其工作方式主要有以下两个区别。

- Cache 由 ASP.NET 控制，当应用程序关闭或者内存不足时会将其删除。这意味着在尝试使用已经缓存的对象之前，必须验证它是否存在。否则，可能会导致空引用的异常。
- Cache 对象是线程安全的。也就是说开发人员不必在添加或者删除缓存项时使用 Lock() 和 Unlock()这样的方法(Application 对象需要管理用户的并发问题)。当然，如果在缓存中存储的对象不是线程安全的，并且有不止一个客户试图同时更改或者使用这个对象，那么仍然可能遇到并发问题。这种情况的解决办法是，可以在缓存中创建缓存项的一个临时副本，然后在 Web 方法中使用这个副本。

使用应用程序缓存的缺点是需要编写实现缓存的代码，不过这些代码可以容易地与 Web 服务方法分开。



应用程序缓存的生命周期依赖于应用程序的生命周期。如果重新启动应用程序，将会重新创建 Cache 对象。

1．向缓存中添加项

在开始使用应用程序缓存之前，首先需要添加要存储的信息。具体方法就像使用 Application 和 Session 对象一样，只需为 Cache 对象指定一个键名就可以向应用程序缓存添加一个新项。

例如，下面的代码演示了使用这种方式存储不同类型的数据。

```
//在应用程序缓存中存储字符串型数据
this.Context.Cache["LastLoginTime"] = DateTime.Now.ToString();
//在应用程序缓存中存储布尔值型数据
this.Context.Cache["IsVIP"] = false;
//在应用程序缓存中存储整型数据
this.Context.Cache["MaxPage"] = 100;
//在应用程序缓存中存储自定义对象
User UserInfo=new User();
this.Context.Cache["User"] = UserInfo;
```

使用这种方法可以很简单地向应用程序缓存中添加需要存储的数据，但是它的缺点是无法控制数据的过期时间。

因此，更好的方法是调用 Cache 类提供的 Insert()方法向 Cache 对象中插入一项作为应用程序缓存。在插入时也可以指定数据的键名和值，例如下面演示了这种方式的代码：

```
//使用 Insert()方法存储整型数据
this.Context.Cache.Insert("OnLineUsers",100);
//使用 Insert()方法存储字符串型数据
this.Context.Cache.Insert("SiteURL", "http://www.itzcn.com");
//使用 Insert()方法存储自定义对象
this.Context.Cache.Insert("User",UserInfo);
```

除此之外，Insert()方法还提供了很多用于控制缓存的重载形式，它们的语法如下：

```
public void Insert(string key, object value);
```

使用 key 指定的键名将 value 值添加到缓存，并使用默认的优先级和过期时间。这种形式与使用“Cache["键"]=值”的语法相同。

```
public void Insert(string key, object value, CacheDependency dependencies);
```

使用 key 指定的键名将 value 值添加到缓存，并使用默认的优先级和过期时间。Dependencies 参数用于指定一个文件或者已缓存项的 CacheDependency 对象，当它发生改变时使缓存项失效。

```
public void Insert(string key, object value, CacheDependency dependencies,
DateTime absoluteExpiration, TimeSpan slidingExpiration);
```

使用 key 指定的键名将 value 值添加到缓存，并使用默认的优先级，还可以指定一个过期时间。这是 Insert()方法最常用的重载形式。

```
public void Insert(string key, object value, CacheDependency dependencies,
DateTime absoluteExpiration, TimeSpan slidingExpiration, CacheItemPriority
priority, CacheItemRemovedCallback onRemoveCallback);
```

这是 Insert()方法最完整的重载形式，可以配置缓存的每个方面，包括过期时间、依赖性和优先级等。此外，还可以通过一个委托来指定删除缓存时要调用的回调函数。

2. 设置项的过期选项

在使用应用程序缓存时，可以通过两种方式来指定缓存项的过期时间。第一种是固定过期时间的方式，即它在超过指定的一段时间之后过期，就像输出缓存的 CacheDuration 一样。这种方式对于数据在固定时间内有效时是最合适的。例如，使用这种方式保留上次查询结果 60 秒。

下面的代码演示了使用这种方式将一个项缓存 10 分钟：

```
this.Context.Cache.Insert("Size", "1000", null, DateTime.Now.AddMinutes(10),
TimeSpan.Zero);
```

第二种方式是指定一个动态过期时间，此时 ASP.NET 将等待一段非活动时间后再处理被忽视的缓存项。例如，指定动态过期时间为 1 小时，那么只有在 1 小时内没有使用的项才会被删除。若某项可能在短短的 20 分钟后失效，但如果对其有持续的需要，那么该项将会被无限期地保留。这种方式对于那些信息总是有效，但是需求可能不高的项非常有用。例如，对于搜索服务可能会检索的历史数据，像调查报告，它们不会因为不再有效而过期。

如果要使用这种方式，需要将 Insert()方法的 absolutExpiration 参数设置为“DateTime.MaxValue”。下面的代码演示了使用这种方式创建一个缓存项，直到该项 10 分钟内没有被使用。

```
this.Context.Cache.Insert("Size", "1000", null, DateTime.MaxValue,
TimeSpan.FromMinutes(10));
```



在创建一个缓存项时不能同时使用上面的两种方式，否则会报出 ArgumentException 异常。

3. 缓存优先级

当我们使用 Insert()方法创建缓存项时，根据使用重载形式的不同，缓存项的优先级可能被赋予了默认值。当然，也可以通过 Insert()方法的 priority 参数来更改为指定优先级。

priority 参数是 CacheItemPriority 类型的一个枚举值，如表 9-1 所示给出了可选的值及其说明。

表 9-1　CacheItemPriority 枚举值

值	说　明
Low	在服务器释放系统内存时，具有该优先级级别的缓存项最有可能被从缓存删除
BelowNormal	在服务器释放系统内存时，具有该优先级级别的缓存项比分配了 Normal 优先级的项更有可能被从缓存删除
Normal	在服务器释放系统内存时，具有该优先级级别的缓存项很有可能被从缓存删除，其被删除的可能性仅次于具有 Low 或 BelowNormal 优先级的那些项。这是默认选项
Default	缓存项优先级的默认值为 System.Web.Caching.CacheItemPriority.Normal
AboveNormal	在服务器释放系统内存时，具有该优先级级别的缓存项被删除的可能性比分配了 Normal 优先级的项要小

续表

值	说　明
High	在服务器释放系统内存时，具有该优先级级别的缓存项最不可能被从缓存删除
NotRemovable	在服务器释放系统内存时，具有该优先级级别的缓存项将不会被自动从缓存删除。但是，具有该优先级级别的项会根据项的固定到期时间与其他项一起被移除



优先级对于缓存清理来说非常重要。当 ASP.NET 检测到服务器内存不足时，它将有选择地删除不经常使用并且优先级最低的项。

4．Cache 类其他成员

在本节前面，我们学习了 Cache 类中 Insert()方法的使用，该方法可以向应用程序中添加缓存，并指定过期时间和缓存优先级等。

此外，在 Cache 类中还提供了很多成员来辅助对缓存的控制。如表 9-2 所示列出了这些成员名称及其说明。

表 9-2　Cache 类其他成员

成员名称	说　明
Add()方法	该方法用于添加一个缓存项，语法与 Insert()方法的第 4 种重载形式相同
Count 属性	该属性返回当前应用程序中缓存项的数量
Get()方法	该方法可以根据 key 来检索指定的缓存项并返回，如果未找到则返回 null
Remove()方法	该方法可以根据 key 来从应用程序中删除一个缓存项

例如，下面的代码使用 Remove()方法实现了清理整个应用程序缓存。

```
public void ClearCache()
{
    //使用 DictionaryEntry 遍历当前的应用程序缓存
    foreach (DictionaryEntry objItem in this.Context.Cache)
    {
        //获取当前缓存项的 key
        string CacheKey = objItem.Key.ToString();
        //删除该缓存项
        this.Context.Cache.Remove(CacheKey);
    }
}
```

9.3.2　实例描述

作为一名站长，网站上线之后，第一个需要考虑的就是如何将网站流量变现，即通过网站实现收入。这个实现方式有多种，广告便是其中之一。

在显示广告之前，我们首先需要对广告的位置有一个规划。例如显示哪种类型的广告，使用什么样的样式，都在哪些位置显示，以及显示的开始和终止时间等。另外，为了使广告不影

响网站的加载速度通常都会保存到缓存中，并在需要的时候进行清理。

下面的实例介绍了如何在 Web 服务中通过应用程序缓存来管理广告位。

9.3.3 实例应用

【例 9-2】 管理缓存的数据

(1) 创建一个名为“AdWebService.asmx”的 Web 服务，并引用以下的命名空间。

```
using System.Web.Caching;
using System.Collections;
```

(2) 在 Web 服务所在的项目下创建一个 Advertisement.cs 文件。它保存的是一个广告实体类，具体成员及其构造函数如下：

```
public class Advertisement
{
   public Advertisement()
   {
   }
   public Advertisement(int id, string title, string type, string style,
DateTime starttime, DateTime endtime, string target)
   {
      this.ID = id;
      this.Title = title;
      this.Type = type;
      this.Style = style;
      this.StartTime = starttime;
      this.EndTime = endtime;
      this.Target = target;
   }
   public int ID;                                          //广告编号
   public string Title { get; set; }                       //广告标题
   public string Type { get; set; }                        //广告类型
   public string Style { get; set; }                       //广告样式
   public DateTime StartTime { get; set; }                 //广告开始时间
   public DateTime EndTime { get; set; }                   //广告结束时间
   public string Target { get; set; }                      //广告投放目标
}
```

如上述代码所示，Advertisement 类包括了 7 个属性以及两个构造函数，其中无参构造函数用于实例化空类，有参构造函数在实例化时可以指定数据。

(3) 返回到 Web 服务中，编写一个 InitAdList()方法对广告位进行初始化，包括指定一些广告位和当前用户，再将它们添加到缓存中。具体实现代码如下所示：

```
 List<Advertisement> list;                         //定义一个广告列表
 public void InitAdList()
 {
   //初始化广告位
```

```
    list = new List<Advertisement>
    {
        new Advertisement(1,"视频推广的首页广告","页首通栏广告","图片",new
         DateTime(2011,1,1),new DateTime(2011,12,31),"整站"),
        new Advertisement(2,"新书推荐广告","页首对联广告","图片",new
         DateTime(2011,5,1),new DateTime(2011,5,7),"学院模块"),
        new Advertisement(3,"赞助商文字广告","页尾版权下方广告","文字",new
         DateTime(2011,1,1),new DateTime(2011,12,31),"首页")
    };
   //将当前用户保存到应用程序缓存中
    this.Context.Cache.Insert("user", "Administraotr", null,
      DateTime.Now.AddMinutes(30), TimeSpan.Zero);
   //将当前的广告列表保存到应用程序缓存中
    this.Context.Cache.Insert("ad", list, null, DateTime.Now.AddHours(1),
      TimeSpan.Zero);
  }
```

(4) 创建一个 InsertAd()方法允许用户向列表中添加一个新的广告。实现代码如下所示：

```
[WebMethod]
public List<Advertisement> InsertAd(Advertisement ad)
{
    list = GetAllAd();              //获取当前广告列表
    list.Add(ad);                   //添加新的广告到列表
    return list;            //返回当前广告列表
}
```

(5) InsertAd()方法中先调用 GetAllAd()方法获取当前保存在缓存中的广告列表。GetAllAd()的具体实现如下：

```
[WebMethod]
public List<Advertisement> GetAllAd()
{
    if (this.Context.Cache["ad"] == null)          //判断缓存中是否有广告列表
    {
        InitAdList();                              //没有就执行初始化
    }
    else
    {
        list = (List<Advertisement>)this.Context.Cache["ad"];   //有就直接返回
    }
    return list;
}
```

(6) 接下来创建一个从缓存中删除指定键的 RemoveAd()，它调用了 Cache 对象的 Remove()方法。代码如下所示：

```
[WebMethod]
public void RemoveAd(string key)
{
    this.Context.Cache.Remove(key);
}
```

(7) 再创建一个 ClearCache()方法来删除当前缓存的所有内容。代码如下所示：

```
[WebMethod]
public void ClearCache()
{
    //使用 DictionaryEntry 来遍历当前的应用程序缓存
    foreach (DictionaryEntry objItem in this.Context.Cache)
    {
        //获取当前缓存项的 key
        string CacheKey = objItem.Key.ToString();
        //删除该缓存项
        this.Context.Cache.Remove(CacheKey);
    }
}
```

(8) 最后创建一个 GetUserName()方法判断缓存中是否存在用户，并返回一个字符串。实现代码如下所示：

```
[WebMethod]
public string GetUserName()
{
    if (this.Context.Cache["user"] == null)            //判断缓存中是否存在用户
    {
        return "未登录用户";                          //没有则返回这个字符串
    }
    else
    {
        return this.Context.Cache["user"].ToString();   //有则返回缓存的用户名
    }
}
```

(9) 到此为止，针对 Web 服务的编码工作就完成了。下面创建一个 ASPX 页面对上面的 Web 服务进行调用。

(10) 添加对 AdWebService.asmx 的 Web 引用，引用名称为“adService”。

(11) 在 ASPX 页面中添加显示当前用户和广告列表的布局。如图 9-6 所示为本实例中使用的布局，其中广告列表使用的是 Repeater 控件。

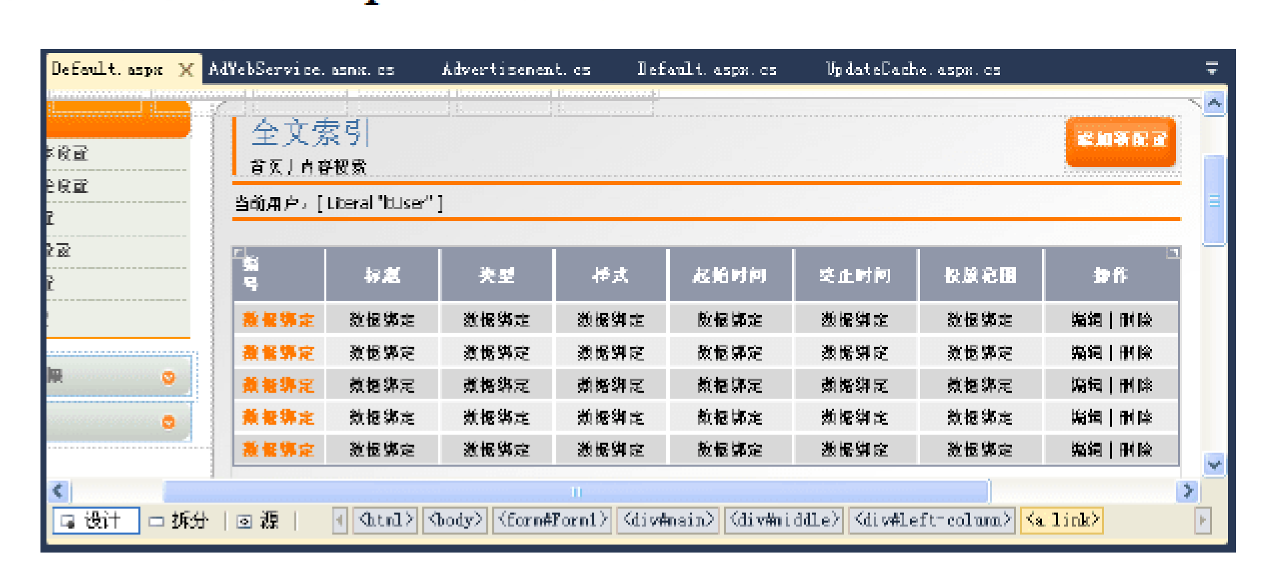


图 9-6 设计布局

(12) 转到后台代码文件，对 Page_Load()方法进行修改。在这里首先实例化一个 Web 服务类，然后获取广告和用户信息并显示到页面上。这部分代码如下所示：

```
    adService.AdWebService ads;
    adService.Advertisement[] list;
    protected void Page_Load(object sender, EventArgs e)
    {
        ads = new adService.AdWebService();                  //实例化 Web 服务类
        if (!IsPostBack)
        {
            list = ads.GetAllAd();                           //获取广告列表
        }
        Repeater1.DataSource = list;                         //指定数据源
        Repeater1.DataBind();                                //绑定显示
        ltUser.Text = ads.GetUserName();                     //显示当前用户
    }
```

(13) 在广告列表的下方制作一个广告位添加的表单，最终效果如图 9-7 所示。

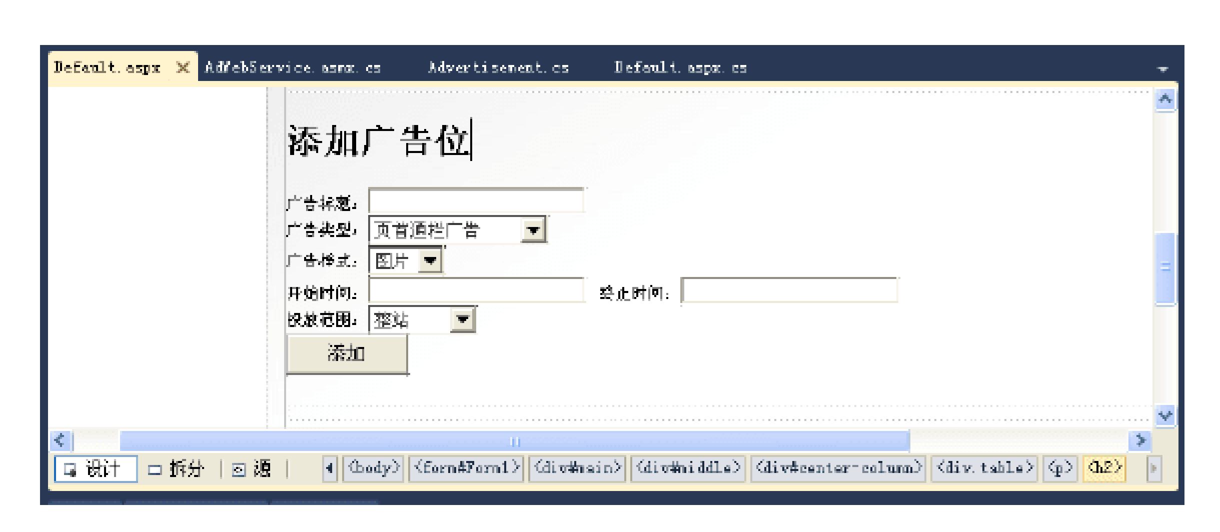


图 9-7　制作添加广告位布局

(14) 进入【添加】按钮的单击事件，编写代码调用 Web 服务的 InsertAd()方法实现添加效果。具体实现代码如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    adService.Advertisement newAD = new adService.Advertisement();
                                                     //创建一个广告类实例
    newAD.ID = 5;
    newAD.Title = txtTitle.Text.Trim();
    newAD.Type = ddlType.SelectedValue;
    newAD.Style = ddlStyle.SelectedValue;
    newAD.StartTime = System.Convert.ToDateTime(txtStartTime.Text.Trim());
    newAD.EndTime = System.Convert.ToDateTime(txtEndTime.Text.Trim());
    newAD.Target = ddlTarget.SelectedValue;
    ads.InsertAd(newAD);                             //调用 InsertAd()方法
    list = ads.GetAllAd();                           //获取最新的列表
    Repeater1.DataSource = list;
    Repeater1.DataBind();                            //重新绑定
                                                     //添加成功
    Page.ClientScript.RegisterStartupScript(this.GetType(), "",
    "<script>alert('添加广告成功！')</script>");
}
```

如上述代码所示，这里使用了 InsertAd()和 GetAllAd()两个 Web 服务方法，并在最后弹出一个提示对话框。

(15) 在本页面再添加一个用于更新缓存的布局，主要包含了两个按钮，如图 9-8 所示。

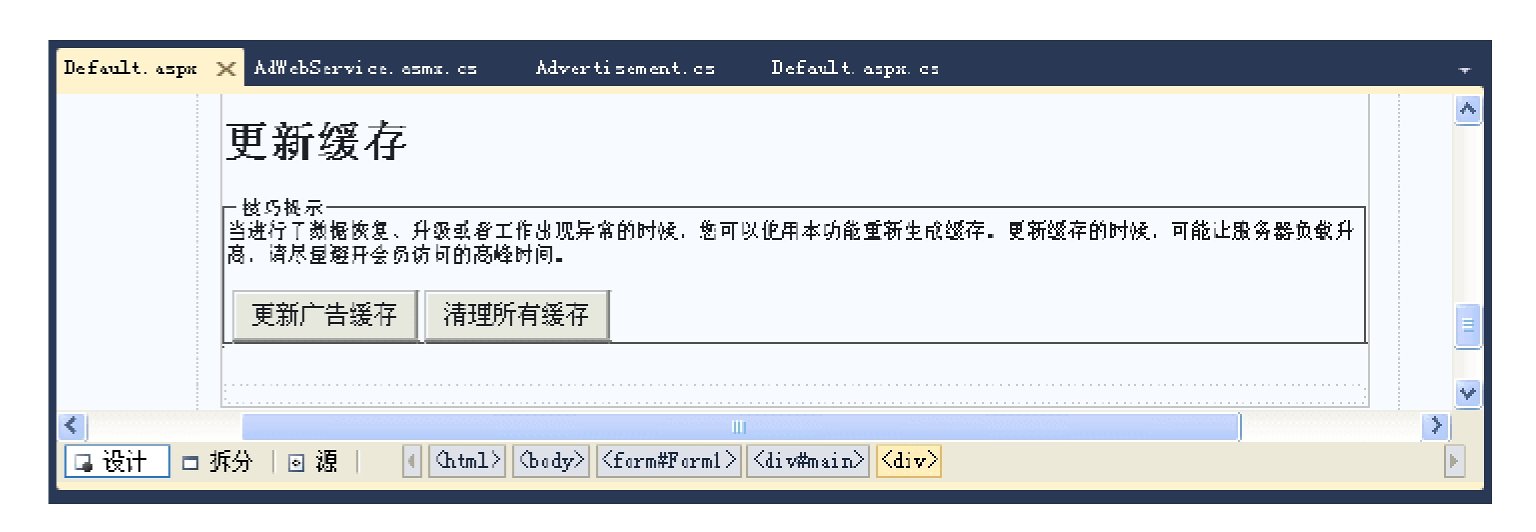


图 9-8　制作更新缓存布局

(16) 更新广告缓存调用的是 RemoveAd()方法，并指定键名来删除广告缓存。实现代码如下所示：

```
protected void btnUpdateAdCache_Click(object sender, EventArgs e)
{
    ads.RemoveAd("ad");
    Page.ClientScript.RegisterStartupScript(this.GetType(), "",
"<script>alert('更新广告缓存成功！')</script>");
}
```

(17) 清理所有缓存则会删除当前所有的缓存项，它调用的是 ClearCache()方法。实现代码如下所示：

```
protected void btnClearAllCache_Click(object sender, EventArgs e)
{
    ads.ClearCache();
    Page.ClientScript.RegisterStartupScript(this.GetType(), "",
"<script>alert('清理缓存成功！')</script>");
}
```

(18) 经过上面创建 Web 服务，添加 Web 服务方法，创建 ASPX 页面，制作布局以及编码工作，至此实例就算完成了。

9.3.4　运行结果

在 Visual Studio 中编译上面创建的 Web 服务项目和 ASP.NET 项目。然后从 ASP.NET 项目中运行页面，此时由于添加了对 Web 服务的引用，Visual Studio 会同时启动 Web 服务项目。

如图 9-9 所示为页面打开后的默认效果，可以看到显示了当前用户和一个广告列表。

在页面下方的添加广告位区域中输入广告位信息，再单击【添加】按钮，会看到提示对话框，如图 9-10 所示。

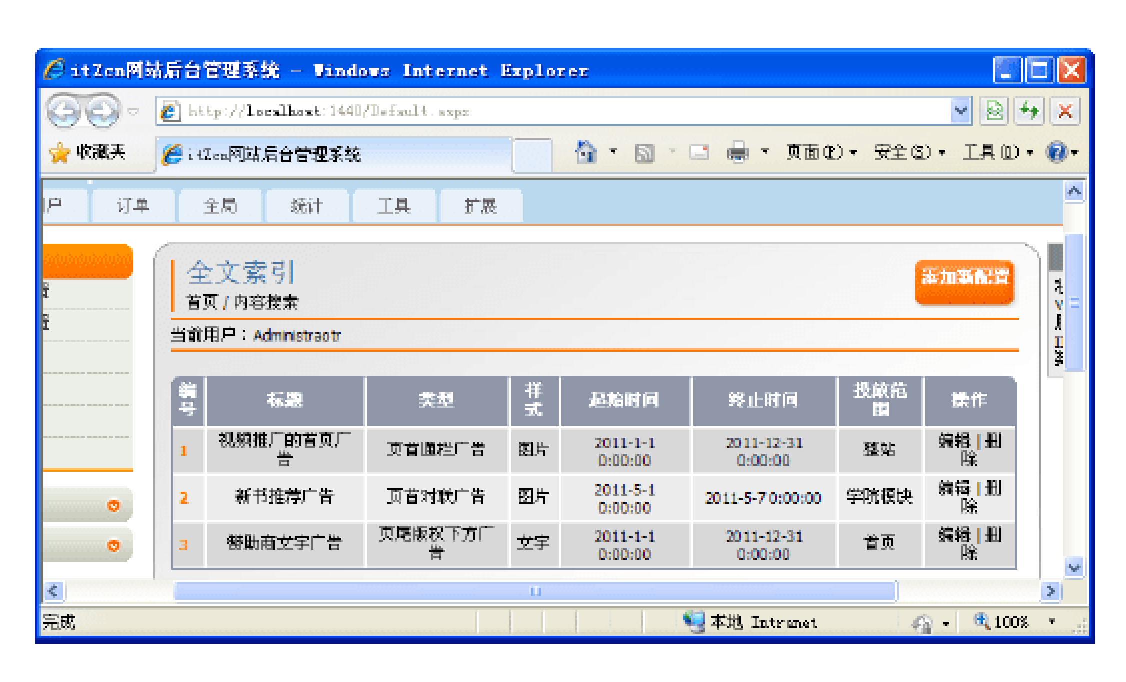


图 9-9　默认效果

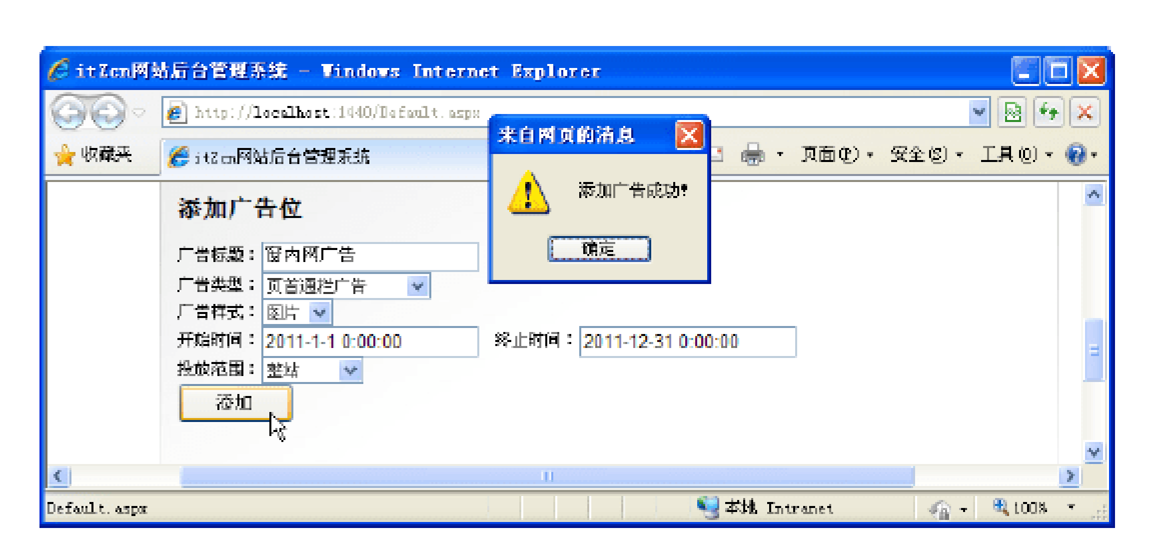


图 9-10　添加广告位效果

单击【确定】按钮返回页面，在广告列表中会看到新增的广告位信息，如图 9-11 所示。



图 9-11　添加后的广告列表

现在测试更新缓存的效果。单击【更新广告缓存】按钮会看到提示对话框，同时页面上的广告列表消失了，如图 9-12 所示。



图 9-12　更新广告缓存效果

单击【清理所有缓存】按钮，此时在缓存中的用户名也被删除。因此，在页面上会看到提示对话框，以及背后的“当前用户：末登录用户”文字，如图 9-13 所示。



图 9-13　清理所有缓存效果

9.3.5　实例分析



源码解析：

在本实例中，主要是通过泛型来管理广告列表，并通过它保存到缓存供客户端调用。但是这里要注意，Web 服务中的 List<Advertisement>类型，在客户端时变成了 Advertisement[]。

另外，在为 Repeater 控件指定数据源并绑定之后，在页面上应该使用“<%# Eval("属性")%>”的方式显示数据。

9.4　解决各个缓存间的依赖性

根据前面的知识可以看到，我们创建的应用程序缓存项主要可以通过 3 种方式来删除。

- 缓存项在超过指定的过期选项后自动失效。
- 使用 Cache.Remove()方法手动删除缓存项。
- 由 ASP.NET 来删除缓存项以释放服务器内存。

除此之外，还可以通过缓存的依赖性来指定它的过期时间。即，在创建缓存项时为其指定一个所依赖的对象，例如一个配置文件。这样，当这个对象发生变化时，依赖它的所有缓存项都将被自动删除。



视频教学：光盘/videos/09/解决缓存间的依赖性.avi　　**长度：**10 分钟

要为一个应用程序缓存项指定依赖性需要使用 CacheDependency 类，该类位于 System.Web.Caching 命名空间。

CacheDependency 类管理缓存项依赖性的方式如下。

(1) 创建一个 CacheDependency 对象，ASP.NET 将开始监视指定的文件和缓存项。

(2) 使用 Insert()方法创建一个应用程序缓存项，并指定为上面创建的 CacheDependency 对象。

(3) 如果 CacheDependency 对象引用的文件更改了，那么 ASP.NET 将删除依赖于它的缓存项。

CacheDependency 类提供了 8 个不同的构造函数，如下所示：

```
public CacheDependency(string filename);
```

初始化一个 CacheDependency 类的新实例监视文件或目录的更改情况。当该资源更改时，缓存的对象将过期，并从缓存中移除。

```
public CacheDependency(string[] filenames);
```

初始化一个 CacheDependency 类的新实例，它监视一组(文件或者目录)路径的更改情况。当这些资源中的任何一个更改时，缓存的对象即过期，并从缓存中移除。

```
public CacheDependency(string filename, DateTime start);
```

初始化一个 CacheDependency 类的新实例，并从指定的时间开始监视文件或目录的更改情况。

```
public CacheDependency(string[] filenames, DateTime start);
```

初始化一个 CacheDependency 类的新实例，并从指定的时间开始监视一组(文件或者目录)路径的更改情况。

```
public CacheDependency(string[] filenames, string[] cachekeys);
```

初始化一个 CacheDependency 类的新实例，它监视一组(文件或者目录)路径或者缓存键的更改情况，还可以同时监视两者。

```
public CacheDependency(string[] filenames, string[] cachekeys, CacheDependency
dependency);
```

初始化一个 CacheDependency 类的新实例，它监视一组(文件或者目录)路径或者缓存键的更改情况，还可以同时监视两者。此外，还可以指定一个它本身所依赖的 CacheDependency 类实例。

```
public CacheDependency(string[] filenames, string[] cachekeys, DateTime
start);
```

初始化一个 CacheDependency 类的新实例，并从指定的时间开始监视一组(文件或者目录)路径或者缓存键的更改情况，还可以同时监视两者。

```
public CacheDependency(string[] filenames, string[] cachekeys, CacheDependency
dependency, DateTime start);
```

初始化一个 CacheDependency 类的新实例，它监视一组(文件或者目录)路径或者缓存键的更改情况，还可以同时监视两者。此外，还可以指定一个它本身所依赖的 CacheDependency 类实例以及开始监视的时间。

使用 CacheDependency 类创建依赖性也非常容易，只需根据要求选择合适的构造函数创建新实例即可。但是要注意，指定文件时需要使用绝对路径。如果我们要使用与 Web 服务相同的目录，则可以使用 Server.MapPath()方法来确定绝对路径。

例如，下面的代码添加一个键名为“DSN”的缓存项，并指定依赖于文件 config.xml。这里的 config.xml 与 Web 服务在相同目录，而且当它发生变化时，DSN 将会从 Cache 中删除。

```
this.Context.Cache.Insert("DSN", ConnectionString,
            new CacheDependency(Server.MapPath("config.xml"))
 );
```

另外，还可以将现有的缓存项作为依赖，从而创建一个新的缓存项。例如，下面的示例代码缓存了 3 个 DataTable 对象，这些对象分别表示产品清单、产品类别和产品价格。其中，产品清单的 DataTable 依赖于产品类别 DataTable 和产品价格 DataTable。如果删除这两个项目中的任何一个，产品清单也将被删除。

```
[WebMethod]
public void InvalidationTest()
{
    //创建 3 个 DataTable
    DataTable dtPrices = new DataTable("Prices");
    DataTable dtCategories = new DataTable("Categories");
    DataTable dtList = new DataTable("Products");
    //先向应用程序中添加两个缓存项
    this.Context.Cache.Insert("Prices", dtPrices);
    this.Context.Cache.Insert("Categories", dtCategories);
    //创建一个字符串数组并指定为上述两个项的键名
    string[] keys={"Prices","Categories"};
    //创建一个依赖项
    CacheDependency dependency=new CacheDependency(null,keys);
    //再添加一个缓存项，并指定它依赖于上面创建的 dependency
    this.Context.Cache.Insert("ProductList", dtList, dependency);
}
```

下面通过一个简单的示例来学习如何使用 CacheDependency 类设置缓存的依赖项。具体步骤如下。

(1) 创建一个名为“DependencyService.asmx”的 Web 服务文件。

(2) 添加对如下两个命名空间的引用。

```
using System.Web.Caching;
using System.IO;
```

(3) 添加一个名为“SetFileDependency()”的 Web 服务方法。在该方法中使用 Cache 对象创建一个应用程序缓存项，并指定它依赖于同目录下的 products.xml 文件。代码如下所示：

```
[WebMethod]
public void SetFileDependency()
{
Product p = new Product();
```

```
    CacheDependency dependency = new
CacheDependency(Server.MapPath("products.xml"));
    this.Context.Cache.Insert("proList", p, dependency);
}
```

如上述代码所示，这里缓存的键名为“proList”，它的值是一个自定义 Product 对象的实例。如下所示为 Product 对象的定义：

```
public class Product
{
    public string Name;
    public decimal Price;
}
```

(4) 下面编写 CheckDependentItem()方法的代码，检索 Cache 对象并判断 proList 缓存项是否有效，并返回结果。代码如下所示：

```
[WebMethod]
public string CheckDependentItem()
{
    if (this.Context.Cache["proList"] == null)
    {
        return "proList 缓存项已经被删除。";
    }
    else
    {
        return "proList 缓存项还有效。";
    }
}
```

(5) 再添加一个 ChangeFile()方法通过程序的方式对 products.xml 文件执行修改操作。代码如下所示：

```
[WebMethod]
public void ChangeFile()
{
    StreamWriter sw = File.CreateText(Server.MapPath("products.xml"));
    sw.Flush();
    sw.Close();
}
```

(6) 至此，对 Web 服务的编写就完成了。下面创建一个 ASPX 页面测试 Web 服务的缓存是否有效。在页面加载的 Page_Load()中编写如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    DependencyService.DependencyService ds = new
    DependencyService.DependencyService();
    Response.Write("创建一个键名为 proList 的应用程序缓存项。<br/>");
    ds.SetFileDependency();
    Response.Write("调用 CheckDependentItem()方法验证是否存在该项。<br/>");
```

```
    string message=ds.CheckDependentItem();
    Response.Write("结果: "+message);
    Response.Write("<hr/><br/>调用 ChangeFile()方法对 products.xml 文件进行修改。
     <br/>");
    ds.ChangeFile();
    Response.Write("再次调用 CheckDependentItem()方法验证是否存在该项。<br/>");
    message=ds.CheckDependentItem();
    Response.Write("结果: " + message);
}
```

(7) 在上述代码中，DependencyService 是对 Web 服务的引用名称。页面运行后的输出结果如图 9-14 所示。

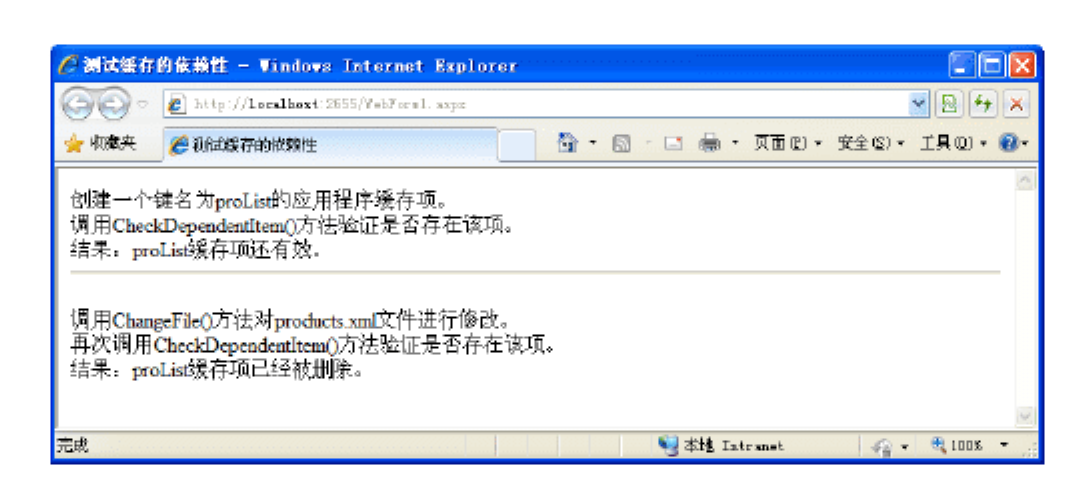


图 9-14 测试缓存依赖性

从图 9-14 所示的输出可以看到，在创建了一个 CacheDependency 对象之后，ASP.NET 就开始监视依赖项。一旦依赖发生了变化，那么 ASP.NET 将移动依赖它的缓存项。

9.5 为删除缓存项指定回调函数

在使用 Insert()方法添加缓存项时，如果为其指定了一个回调函数。那么当从缓存中删除该项时，ASP.NET 会先调用指定的回调函数，再执行删除操作。

因此，我们使用这种机制可以对缓存项执行清理(删除相关的资源)或者跟踪调试等操作。

 **视频教学：光盘/videos/09/指定回调函数.avi** 长度：11 分钟

9.5.1 基础知识——缓存回调函数

为了能够指定缓存项的删除回调函数，首先必须使用 Insert()方法的如下重载形式添加缓存项。

```
public void Insert(string key, object value, CacheDependency dependencies,
DateTime absoluteExpiration, TimeSpan slidingExpiration, CacheItemPriority
priority, CacheItemRemovedCallback onRemoveCallback);
```

这里的 onRemoveCallback 参数是一个 CacheItemRemovedCallback 类型的委托，它的值应该是一个不带括号的函数名称(例如，应该是 callback 而不是 callback())。

如下所示为这个回调函数的声明格式：

```
delegate void CacheItemRemovedCallback(string key, object value,
CacheItemRemovedReason reason);
```

可以看到，ASP.NET 会向回调函数中传递被删除缓存项的键名和值，以及删除该项的原因。其中，reason 参数表示删除的原因，它是一个 CacheItemRemovedReason 类型的枚举值，可选项有如下几个。

- DependencyChanged：从缓存移除该项的原因是与之关联的缓存依赖项已更改。
- Expired：从缓存移除该项的原因是它已过期。
- Removed：该项是通过指定键名并调用 Remove()方法从缓存中移除的。
- Underused：之所以从缓存中移除该项，是因为系统要通过移除该项来释放内存。

例如，下面的代码演示了回调函数的使用方法。

```
//定义一个委托类型
CacheItemRemovedCallback onRemove = null;
//回调函数
public void RemovedCallBack(string key, object value, CacheItemRemovedReason
reason)
{
    //在这里根据删除原因进行相应的处理
}
 public void SetCacheItem()
 {
    //指定委托的回调函数
    onRemove = new CacheItemRemovedCallback(this.RemovedCallBack);
    //使用 Insert()方法添加一个缓存项，并指定回调函数
    if (this.Context.Cache["key"] == null)
        this.Context.Cache.Insert("key", "value", null,
            DateTime.Now.AddSeconds(60), TimeSpan.Zero,
            CacheItemPriority.High,
            onRemove
            );
}
```

9.5.2　实例描述

软件设计借鉴了硬件设计中引入缓存机制以改善整个系统的性能，尤其是对于一个数据库驱动的 Web 应用程序而言，缓存的利用是不可或缺的。毕竟，数据库查询可能是整个 Web 站点中调用最频繁但同时又是执行最缓慢的操作之一，我们不能让它老迈的双腿拖住我们前进的征程。缓存机制正是解决这一缺陷的加速器。

下面让我们一起来看看 Web 服务中应用程序缓存的强大功能，以及了解它是如何通过回调函数来跟踪缓存的删除。

9.5.3　实例应用

【例 9-3】 为删除缓存项指定回调函数

(1) 在 Visual Studio 中为实例创建一个 Web 服务文件。

(2) 使用如下代码添加对命名空间的引用。

```
using System.IO;
using System.Collections;
using System.Web.Caching;
```

(3) 首先添加一个 AddCacheItems()方法，它向 Cache 对象中添加了 3 个缓存项，并指定了不同的过期时间。当从 Cache 对象中删除它们时，都会调用 onRemoveCacheItem()方法，具体实现代码如下所示：

```
//向 Cache 对象中添加几个数据项，然后返回这些数据项的名称和值
[WebMethod(Description = "向 Cache 对象中添加几个数据项，然后返回这些数据项的名称和值
")]
public string[] AddCacheItems()
{
    //创建代理对象实例
    onRemoveCacheItem = new CacheItemRemovedCallback(this.RemoveCallBack);
    if (this.Context.Cache["Expired30s"] == null)
        this.Context.Cache.Insert("Expired30s",
            "该缓存项在 30 秒后失效", null,
            DateTime.Now.AddSeconds(30), TimeSpan.Zero,
            CacheItemPriority.High,
            onRemoveCacheItem
            );
    //添加一个依赖于 Expried30s 的缓存项
    if (this.Context.Cache["DependExpired30s"] == null)
    {
        string[] dependencyKey = new string[1];
        dependencyKey[0] = "Expired30s";
        CacheDependency dependency = new CacheDependency(null, dependencyKey);
    this.Context.Cache.Insert(
        "DenpendExpired30s",
        "该缓存项依赖于 Expired30s",
        dependency,
        DateTime.Now.AddMinutes(30), TimeSpan.Zero,
        CacheItemPriority.High,
        onRemoveCacheItem
        );
}
//添加一个依赖于文件的缓存项
if (this.Context.Cache["DependFile"] == null)
{
    CacheDependency dependency = new CacheDependency("e:\\log.txt");
    this.Context.Cache.Insert("DenpendFile", "该缓存项依赖于文件 E:\\log.txt,
     当文件发生变化时它将被删除。",
            dependency,
```

```
            DateTime.Now.AddMinutes(30), TimeSpan.Zero,
            CacheItemPriority.High,
            onRemoveCacheItem
            );
    }
    //返回 Cache 对象中的数据项
    string[] items = new string[this.Context.Cache.Count];
    int i = 0;
    foreach (DictionaryEntry objItem in this.Context.Cache)
    {
        string strName = objItem.Key.ToString();
        items[i++] = strName + ":" + this.Context.Cache[strName];
    }
    return items;
}
```

(4) 创建 AddCacheItems()方法所需的回调函数 onRemoveCacheItem。在这个回调函数中将记录每一次的删除操作，实现代码如下所示：

```
private CacheItemRemovedCallback onRemoveCacheItem = null;
//定义一个回调函数，在这个回调函数中将记录每一次删除操作，这些记录将保存在 Application 对
象中
private void RemoveCallBack(string name, object thevalue,
CacheItemRemovedReason reson)
{
    ArrayList logs;
    if (Application["logs"] == null)
        logs = new ArrayList();
    else
        logs = (ArrayList)(Application["logs"]);
    string log = "你在[" + DateTime.Now.ToString() + "]时执行了删除操作。删除目标：
(" + name + "： " + thevalue + ")，删除原因： " + reson.ToString();
    logs.Add(log);
    Application["logs"] = logs;
}
```

(5) 编写一个 GetLogs()方法获取回调函数的删除操作，实现代码如下所示：

```
//获取回调函数记录的删除操作
[WebMethod(Description = "获取回调函数记录的删除操作")]
public string[] GetLogs()
{
    if (Application["logs"] == null)
        return null;
    ArrayList logs = (ArrayList)(Application["logs"]);
    string[] thelogs = new string[logs.Count];
    for (int i = 0; i < logs.Count; i++) thelogs[i] = (string)(logs[i]);
    return thelogs;
}
```

(6) 接下来创建一个 SaveLogs()方法把删除记录保存到一个外部文件中，实现代码如下所示：

```
//把删除记录保存到文件 log.txt 中
[WebMethod(Description = "把删除的记录保存到 log.txt 中，这将导致 DependFile 缓存项的删除")]
public string SaveLogs()
{
    ArrayList logs = (ArrayList)(Application["logs"]);
    if (logs == null) return "没有删除记录";
    StreamWriter sw = new StreamWriter("e:\\log.txt", true);
    for (int i = 0; i < logs.Count; i++)
    {
        sw.WriteLine((string)logs[i]);
    }
    sw.Close();
    return "删除记录保存成功";
}
```

(7) 至此，关于 Web 服务的操作缓存项的代码就编写完成了。

9.5.4 运行结果

现在通过浏览器来测试上面创建的 Web 服务，如图 9-15 所示为打开后的页面。

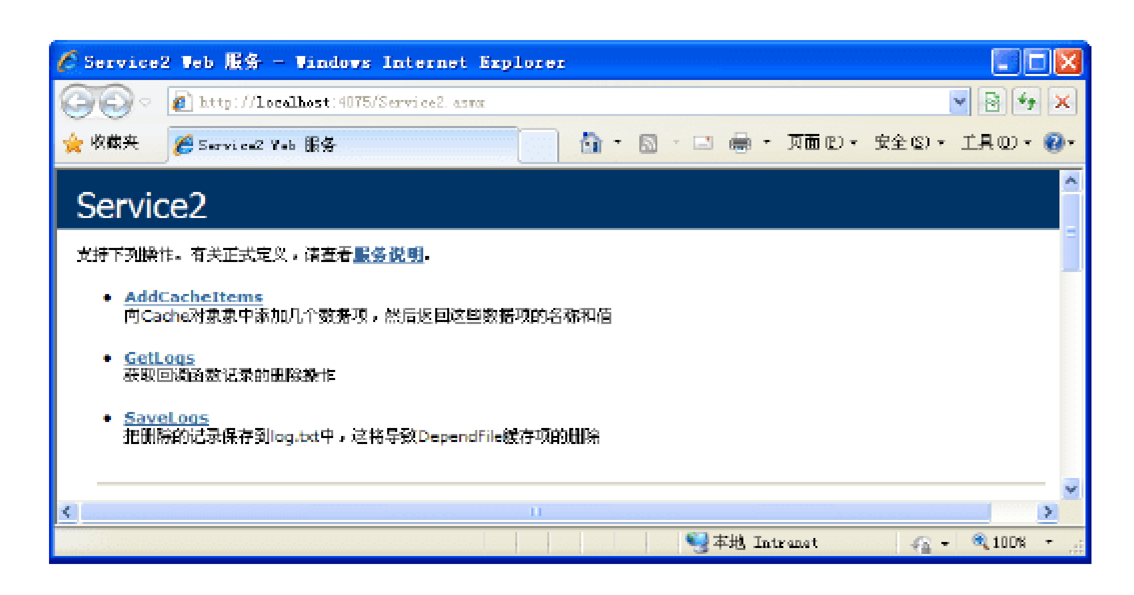


图 9-15 浏览 Web 服务

这里先对 AddCacheItems()方法进行测试，单击该链接并单击【调用】按钮会看到该方法返回 Cache 对象中的所有缓存项，如图 9-16 所示。

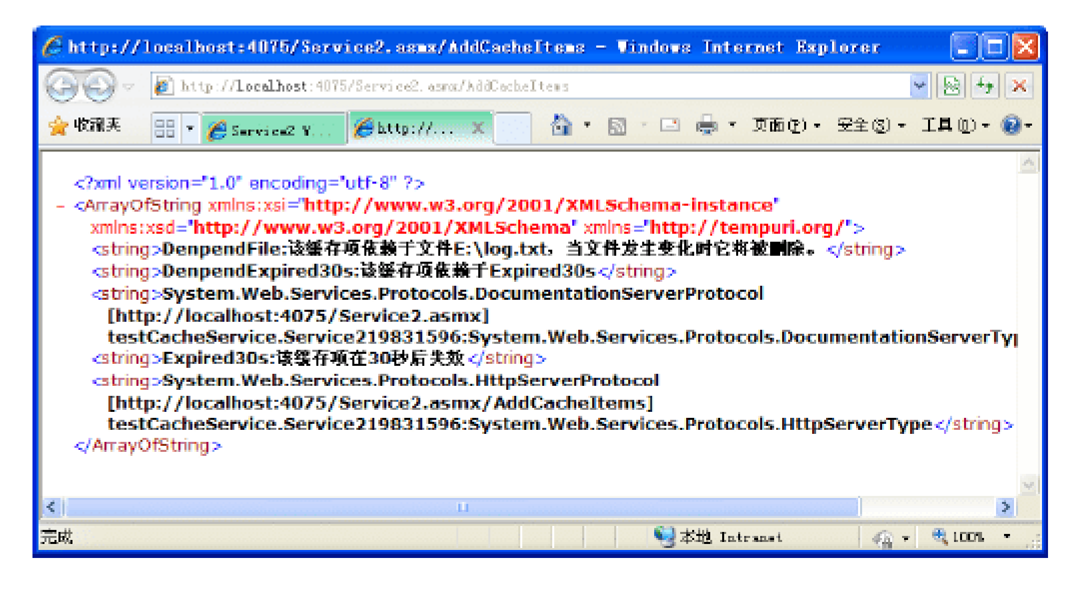


图 9-16 保存在 Cache 对象中的缓存项

从图 9-16 中可以看到，除了保存的 3 个缓存项外，ASP.NET 还在 Cache 对象中保存了其他很多数据。

下面调用 GetLogs()方法查看 onRemoveCacheItem()方法记录的删除操作。因为程序开始时，没有缓存项被删除，所以返回结果为空，如图 9-17 所示。

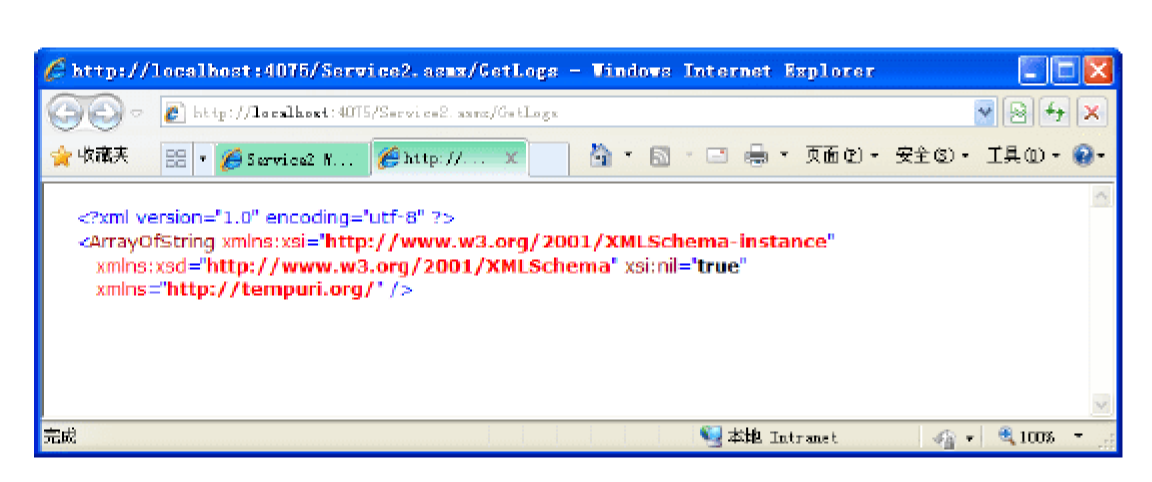


图 9-17　没有缓存项被删除时

此时，如果等待 30 秒之后，这时第 1 个缓存项 ExpiredAfter30s 将会被从 Cache 对象中删除，ASP.NET 将会自动调用指定的 onRemoveCacheItem()方法。又因为 DependExpiredAfter30s 缓存项依赖于 Expired30s 缓存项，所以 ExpiredAfter30s 的删除将会导致 DependExpiredAfter30s 也被删除，如图 9-18 所示。

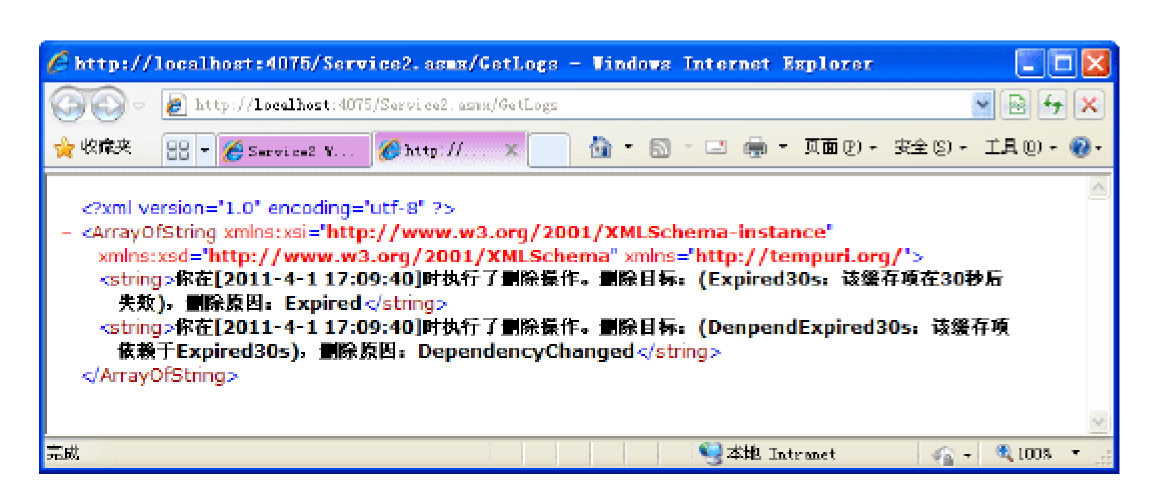


图 9-18　ExpiredAfter30s 导致的 DependExpiredAfter30s 被删除

从图 9-18 中可以看到，ExpiredAfter30s 缓存项的删除原因是 Expired，即过了有效期限；DependExpiredAfter30s 缓存项的删除原因是 DependencyChanged，即它所依赖的对象发生了改变。

最后，为了观察 DependFile 缓存项被删除的情况，在这里可以调用 SaveLogs()方法。该方法将会把 onRemoveCacheItem 记录的删除操作保存到文件“E:\log.txt”中，而 DependFile 缓存项又依赖于这个文件。所以这个方法将会导致 DenpendFile 缓存项被删除，如图 9-19 所示为调用 SaveLogs()方法的结果。

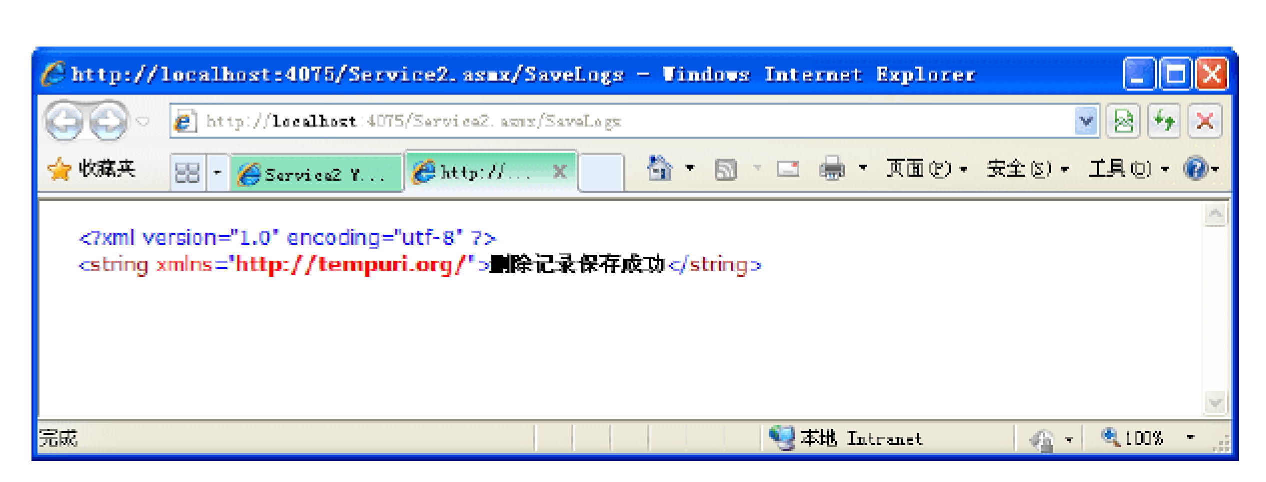


图 9-19　调用 SaveLogs()方法结果

在调用 SaveLogs()方法之后，如果用户再次调用 GetLogs()方法将会看到如图 9-20 所示的结果。

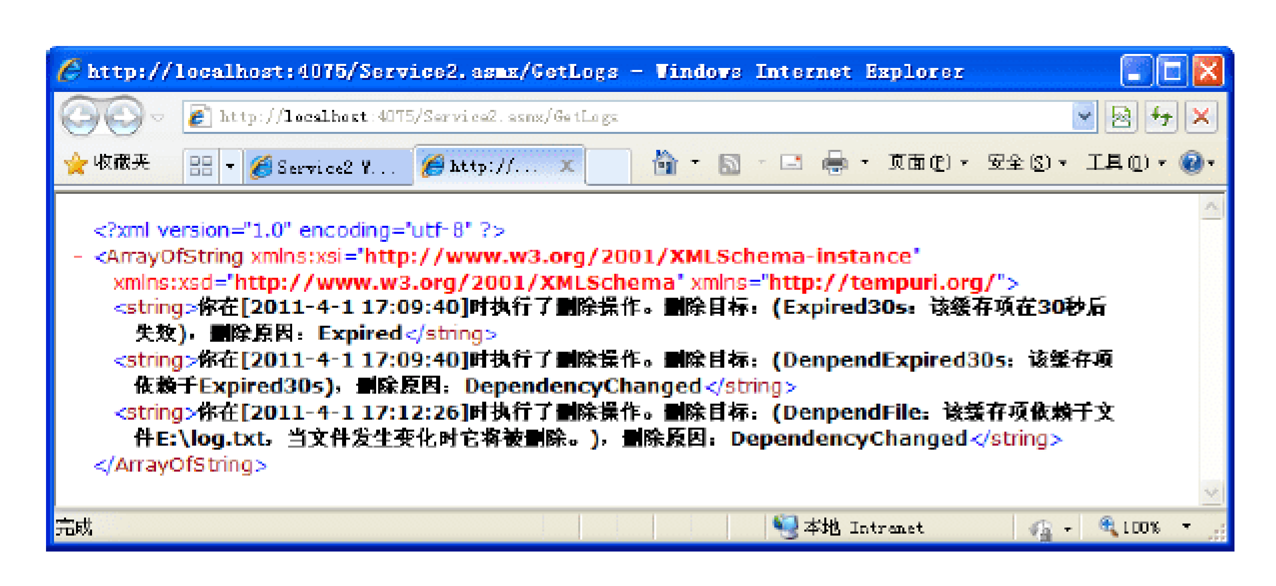


图 9-20　保存文件导致了 DenpendFile 缓存项被删除

从图 9-20 中可以看到，对文件的保存操作导致了 DependFile 缓存项的删除，它的删除原因为 DependencyChanged。

9.5.5　实例分析

**源码解析：**

本实例的重点是使用缓存项的删除回调函数来跟踪缓存项的删除时间以及删除原因。

在本实例中，虽然创建了 3 个缓存项，但是只需要定义一个回调函数即可。第 1 个缓存项为固定过期时间，即 30 秒；第 2 个缓存项的过期时间依赖于第 1 个缓存项；第 3 个缓存项的过期时间依赖于一个外部文件。

9.6　使用缓存时的注意事项

虽然缓存在 ASP.NET 中的使用非常简单，而且缓存能有效提高程序的性能。但是，这并不意味着缓存使用得越多就越好。

因为，如果我们在缓存中保存了太多的信息，那么 ASP.NET 将会自动丢弃一些多余的信息。这与使用会话状态(在超时之前，信息可以一直保留)或者应用程序状态(信息在应用程序的生命周期中一直保留)不同。因此在缓存过满时，可能会挤出其他更重要的信息，也会降低应用程序缓存的效率和性能。

可见，缓存是一把双刃剑，只有合理使用才能发挥作用。本节我们就来了解一些使用缓存时需要注意的事项。

**视频教学：**光盘/videos/09/使用缓存的注意事项.avi　　长度：6 分钟

使用缓存的主要目的就是为了提高应用程序的性能，在使用缓存时用户需要注意以下事项。

(1) 不要缓存大量的数据。因为缓存是在内存中保存数据，如果数据量很大将会占据大量的内存，这将会严重影响程序的性能。

(2) 用户应该只缓存那些变化不太频繁的数据。因为缓存的主要作用就是把数据暂时保存

在内存中，当下次请求这些数据时，就不用再次执行创建过程，这样就可以提高性能。因此，如果数据频繁地变化，则内存中缓存的数据就会频繁地失效，就无法起到应有的作用，反而会降低程序的性能。

(3) 尽量不要缓存特定于用户的数据。因为如果用户数量很大，将会造成缓存的数据量很大，这将得不偿失。而且，缓存特定于用户的私有数据有可能会造成安全隐患。

(4) 用户不仅可以缓存应用程序的响应数据，也可以缓存其他的应用程序数据或者资源，以此来提高性能。例如，用户可以缓存一个数据集对象，这样后续的查询就可以直接在内存中进行，这可以明显地提高程序的性能。

(5) 如果用户能够预料到下一步操作所需的数据，也可以把这部分数据缓存。例如，客户要查找并下载一篇文章，在查找结束后就可以提前把这篇文章加载到内存中。这样，当客户下载时，就可以直接从内存中获取它。

(6) 用户在客户端和服务器端都可以使用缓存。例如，假设存在一个访问数据库的 Web 服务，则用户既可以充分利用数据库服务器的缓存机制，也可以在 Web 服务中使用缓存，这样当遇到相同的调用时，就可以直接返回结果而不用再去查询数据库。最后，还可以在客户端把 Web 服务的返回结果缓存，这样就不用去频繁地通过网络来调用 Web 服务。

(7) 在使用缓存时，用户需要注意的一个关键问题是缓存的有效期。因为缓存中的数据不可能永远有效，所以用户需要确定一个比较合适的更新缓存的时间间隔。

(8) 当在客户端使用缓存时，用户应该建立一种通知客户端数据失效的机制。这可以通过在返回的数据中包含有效期信息来实现。例如，用户可以在 SOAP 消息中添加一个表示数据有效期的字段。

(9) 因为缓存中的数据经常发生变化，所以用户一定不要依赖于缓存中的数据，而是要随时判断缓存中是否存在所需的数据，并根据判断结果进行相应的处理。

9.7　了解事务的并发机制

事务是工作的逻辑单元，一个事务由一个或多个完成一组相关行为的语句组成。管理事务就是确保这一组语句所做的操作要么完全成功地执行，完成整个工作单元的操作；要么一点也不执行。

事务通常与数据库的关系非常密切，因此下面结合数据库来对事务进行解释。

 **视频教学：**光盘/videos/09/了解事务的并发机制.avi　　长度：10 分钟

下面以银行转账过程为例说明事务的特性。将一笔资金从一个账户转移到另一个账户，对于顾客来说，发生的转账只是一个运作，但在数据库系统中是由转出和转入等几个操作组成的。显然，这些操作要么全都发生，要么由于出错而全不发生，这是最基本的一点。

如果数据库只完成了部分操作，比方说只执行了转出或转入，那么就有可能出现某个账户上平白地少了或者多出一些资金的情况。

所以，需要事务机制来保证这些操作序列的逻辑整体性。为了便于从形式上说明银行转账问题，我们假定事务采用以下两种操作来访问数据。

- Read(x)：从数据库发送数据项 x 到事务工作区。
- Write(x)：从事务工作区把数据项 x 传回数据库。

假如，现在要从账户 AccountA 过户 500 元到账户 AccountB，可用下列形式定义转账事务：

```
Read(AccountA);                    //读取 AccountA 的资金
AccountA=AccountA-500;             //改变 AccountA 的资金，减少 500
Write(AccountA);                   //显示 AccountA 的当前余额
Read(AccountB)                     //读取 AccountB 的资金
AccountB=AccountB+500;             //改变 AccountB 的资金，增加 500
Write(AccountB);                   //显示 AccountB 的资金余额
```

事务的主要作用就是保证数据库的完整性。因此，从保证数据库完整性出发，我们要求数据库管理系统维护事务的几个性质：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)，简称为 ACID，下面分别对它们加以讲述。

1．原子性

事务的原子性是指事务中包含的所有操作要么全做，要么全不做。只有在所有的语句都正确完成的情况下，事务才能完成并把结果应用于数据库。也就是说：事务的所有活动在数据库中要么全部反映，要么全部不反映，以保证数据库是一致的。

例如，转账事务在 Write(AccountA)操作执行完之后、Write(AccountB)操作执行之前，数据库反映出来的结果为：账户 AccountA 少了 500 元，而账户 AccountB 还未增加 500 元，此时“账户 AccountA＋账户 AccountB”的总额少了 500 元。所以事务执行到某个时刻数据库是不一致的，但是事务执行完成后，这个暂时的内部不一致状态就会被账户 AccountB 增加 500 元所代替。

保证原子性的基本思路如下：对于事务要执行写操作的数据项，数据库系统会在磁盘上记录其旧值，如果事务没有完成，旧值被恢复，好像事务从未执行过。

2．一致性

事务开始之前，数据库处于一致性的状态；事务结束后，数据库必须仍处于一致性状态。

以转账事务为例，尽管事务执行完成后，账户 AccountA 和 AccountB 的状态多种多样，但一致性要求事务的执行不应改变账户 AccountA 和 AccountB 的总额，即转入和转出应该是平衡的。如果没有这种一致性要求，转账过程中就会发生钱无中生有或不翼而飞的现象。事务应该把数据库从一个一致状态转换到另一个一致状态。

3．隔离性

在事务的处理过程中暂时不一致的数据不能被其他事务应用，直到数据再次一致。换句话说，当事务使数据不一致时，其他事务将不能访问该事务中不一致的数据。

例如，转账事务在执行完 Write(AccountA)之后、执行 Write(AccountB)之前，数据库中账户 AccountA 中少了 500 元，账户 AccountB 并没有增加 500 元，此时是不一致的。如果另外一个事务基于此不一致状态开始为每个账户结算利息的话，那么显然银行会少支付由 500 元产生的利息。

4．持久性

一个事务成功完成后，它对数据库的改变就被保护起来，即便是在系统遇到故障的情况下也不会丢失。例如，如果转账事务执行完毕，意味着资金的流转已经发生了，那么用户无论何时都应该能够对此加以验证，系统就必须保证出现任何系统故障都不会丢失与这次转账相关的数据。

事务一旦发生任何问题，整个事务就重新开始。数据库也返回到事务开始前的状态。所发生的任何行为都会被取消，数据也恢复到其原始状态。事务要成功完成的话，所有的变化都在执行。在整个过程中，无论事务是否完成或者是否必须重新开始，事务总是确保数据库的完整性。

9.8　为 Web 服务启用事务功能

通过对上一节的学习，我们了解了事务的概念以及它的 4 个特性。为了在 Web 服务中使用事务功能，我们需要为 Web 服务方法的 WebMthod 属性添加 TransactionOption 选项。之后事务的提交与回滚完全由 ASP.NET 来完成，无须用户控制。

下面将介绍事务的具体使用方法。


视频教学：光盘/videos/09/启用事务功能.avi　　　　长度：4 分钟

TransactionOption 选项的值是一个 TransactionOption 类型的枚举值，这些值位于 System.EnterpriseServices 命名空间。

因此，用户在使用时首先需要添加对 System.EnterpriseServices 的引用。TransactionOption 类型的各个值如下。

- Disabled：禁用事务，表示方法不在事务的范围内运行，这是该选项的默认值。
- NotSupported：表示方法不支持事务功能，将在没有事务的情况下运行。
- Required：表示方法需要在事务内运行，此项与 RequiredNew 选项作用相同将创建一个新的事务。
- RequiresNew：表示方法需要新的事务才能运行。即每次请求该方法时，都将在一个新创建的事务内运行它。
- Supported：表示方法支持事务，但是不在事务的范围内运行。

在默认情况下，Web 服务方法是禁用事务的。因此，如果决定在 Web 服务中使用事务，那么 Web 服务的方法只能作为事务中的一个根对象来参与事务。这也意味着，Web 服务方法可能会调用其他支持事务的对象，但是它本身不会作为另一个对象运行事务的一部分来调用。存在这种限制的原因是由于 HTTP 协议是无状态的。

由此产生的结果是，TransactionOption 的 Required 和 RequiredNew 是等价的(都声明一个运行新事务的 RequiredNew 方法)，Disabled、NotSupported 和 Supported 是等价的(都将禁用 Web 服务方法的事务)。

例如，下面的代码指定 DeleteArticle()方法需要在一个新事务中运行，并且该方法将对数

据库进行操作。

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public int DeleteArticle(int artID)
{
    //指定一个标准可以执行的 SQL 语句
    string delSQL = "delete from articles where id=" + artID;
    //指定一个不存在表执行删除操作，因此它将产生一个异常
    string errorDelSQL = "delete from category where id=" + artID;
    //指定数据库连接字符串
    SqlConnection con = new SqlConnection("server=.;user id=sa;database=cms");
    //准备执行的 SqlCommand 对象
    SqlCommand deleteCmd = new SqlCommand(delSQL, con);
    SqlCommand ErrorCmd = new SqlCommand(errorDelSQL, con);
    //连接到数据库
    deleteCmd.Connection.Open();
    //执行删除命令
    deleteCmd.ExecuteNonQuery();
    //执行不存在的删除操作，将会产生异常并导致事务回滚，因此上面的删除操作也被回滚
    int cmdResult = ErrorCmd.ExecuteNonQuery();
    //关闭数据库连接
    con.Close();
    return cmdResult;
}
```

上面的 DeleteArticle()方法因为使用 TransactionOption 选项启用了事务，所以当该方法的执行过程产生任何异常时，ASP.NET 都会自动终止事务。同时，该方法的所有操作都位于一个事务内，所以当出现异常时，该方法内的所有操作都将被取消和回滚。除非该方法内的所有操作都正常完成，此时该方法对应的事务才会被提示，对数据库的改变才会真正产生作用。

9.9　常见问题解答

9.9.1　设置 Web 服务的响应时间和数据传输长度



设置 Web 服务的响应时间和数据传输长度。

网络课堂：http://bbs.itzcn.com/thread-15744-1-1.html

我用 ASP.NET 架设了一个 Web 服务项目，当上传 1 万多条记录时，如果超过 1 分钟，则出现“操作超时”的提示，如果上传 2 万条记录，会提示超出最大长度。

请问，该如何设置 Web 服务的响应时间和传输最大长度？

【解决办法】

你可以为 Web 方法的 CacheDuration 选项设置一个比较大的值，它以秒为单位。另外，也

可以在站点的 Web.config 中进行修改，这部分代码如下所示：

```
<configuration>
     <system.web>
        <httpRuntime   maxRequestLength= "100000 " executionTimeout= "9000 "/>
     </system.web>
</configuration>
```

9.9.2　Web 服务中更新缓存问题



Web 服务中更新缓存问题。
网络课堂：http://bbs.itzcn.com/thread-15745-1-1.html

我用 CacheDuration 设置了缓存，但如果在这个时间内有内容进行了更新。那用户应该如何获取最新的数据？

【解决办法】

你的问题可以通过缓存的依赖项来解决。下面给出一段示例代码：

```
[WebMethod(CacheDuration=600)]
public string HelloWorld() {
HttpContext.Current.Cache.Insert("test", string.Empty);
HttpContext.Current.Response.AddCacheItemDependency("test");
return DateTime.Now.ToString();
}
```

这样一来，如果在其他地方更新数据了，那么该方法需要进行更新时只需使用“HttpContext.Current.Cache.Remove("test");”语句即可；然后这个 HelloWorld()方法会重新执行，产生新的缓存。

9.10　习　　题

一、填空题

(1) 在 ASP.NET 中可以通过__________来缓存页面中的一部分内容。

(2) Cache 类属于__________命名空间，在 Web 服务中可以通过__________来访问这个类的对象实例。

(3) 应用程序缓存的生命周期依赖于__________。

(4) 在 Cache 类中使用__________属性返回当前应用程序中缓存项的数量，使用__________方法删除一个缓存项。

(5) 在 Insert()方法中使用__________参数指定缓存项的删除回调函数。

(6) 假设有一个名为 RemoveItem 的回调函数，补充下面的代码，使它具有高优先级，并

带有回调函数。

```
   CacheItemRemovedCallback CallBack = new
CacheItemRemovedCallback(RemoveItem);
   this.Context.Cache.Insert("key", "value", null,
      DateTime.Now.AddSeconds(60), TimeSpan.Zero,
      __________,
      __________
      );
```

(7) 事务必须具备的四个属性为：原子性、一致性、__________和持久性。

(8) 如果将 TransactionOption 选项的值设置为__________来指定 Web 服务方法需要新的事务才能运行。

二、选择题

(1) 我们要缓存一个页面的内容应该使用__________。

A. 整页缓存　B. 单页缓存　C. 输出缓存　D. 部分页缓存

(2) 下面关于输出缓存的描述不正确的是__________。

A. 能够提高页面的响应速度

B. 超过指定时间之后自动过期，不用手动控制

C. 在 Web 服务中使用 CacheDuration 选项来开启

D. 可以缓存任何类型的数据，例如 DataSet

(3) 假设要创建一个缓存 1 分钟的输出缓存项，应该使用__________代码。

A. CacheDuration=1　B. CacheDuration=6000

C. CacheDuration=600　D. CacheDuration=60

(4) 下面关于输出缓存与应用程序缓存的描述，不正确的是__________。

A. 前者存储 Web 服务方法的结果，后者可以存储任何类型的信息

B. 前者使用 CacheDuration 开启，后者使用 Cache 类开启

C. 前者的缓存时间不可调整，后者可手动控制

D. 两者都可以使用缓存项的回调函数

(5) 如果要向应用程序中添加一个缓存项，下面代码不正确的是__________。

A. this.Context.Cache["Time"] = "2011-11-11"

B. this.Context.Cache.Insert("Time ", "2011-11-11");

C. this.Context.Cache.Add("Time ", "2011-11-11");

D. this.Context.Cache.Insert("Size", "1000", null, DateTime.Now.AddMinutes(10), TimeSpan.Zero);

(6) 在使用 priority 参数配置缓存的优先级时，它的默认值是__________。

A. BelowNormal　B. Normal

C. AboveNormal　D. High

(7) 要在 Web 服务中使用事务功能，必须引用的命名空间是__________。

A. System.EnterpriseServices

B. System.Web.Transaction

C. System.Web.Caching

D. System.Data.Transaction

三、上机练习

上机练习 1： 利用输出缓存保存数据。

本次上机练习要求读者使用 Web 服务的输出缓存来完成。实现将一个 7 位数字缓存 1 天，然后在页面调用并显示这些数字。最终运行效果如图 9-21 所示。

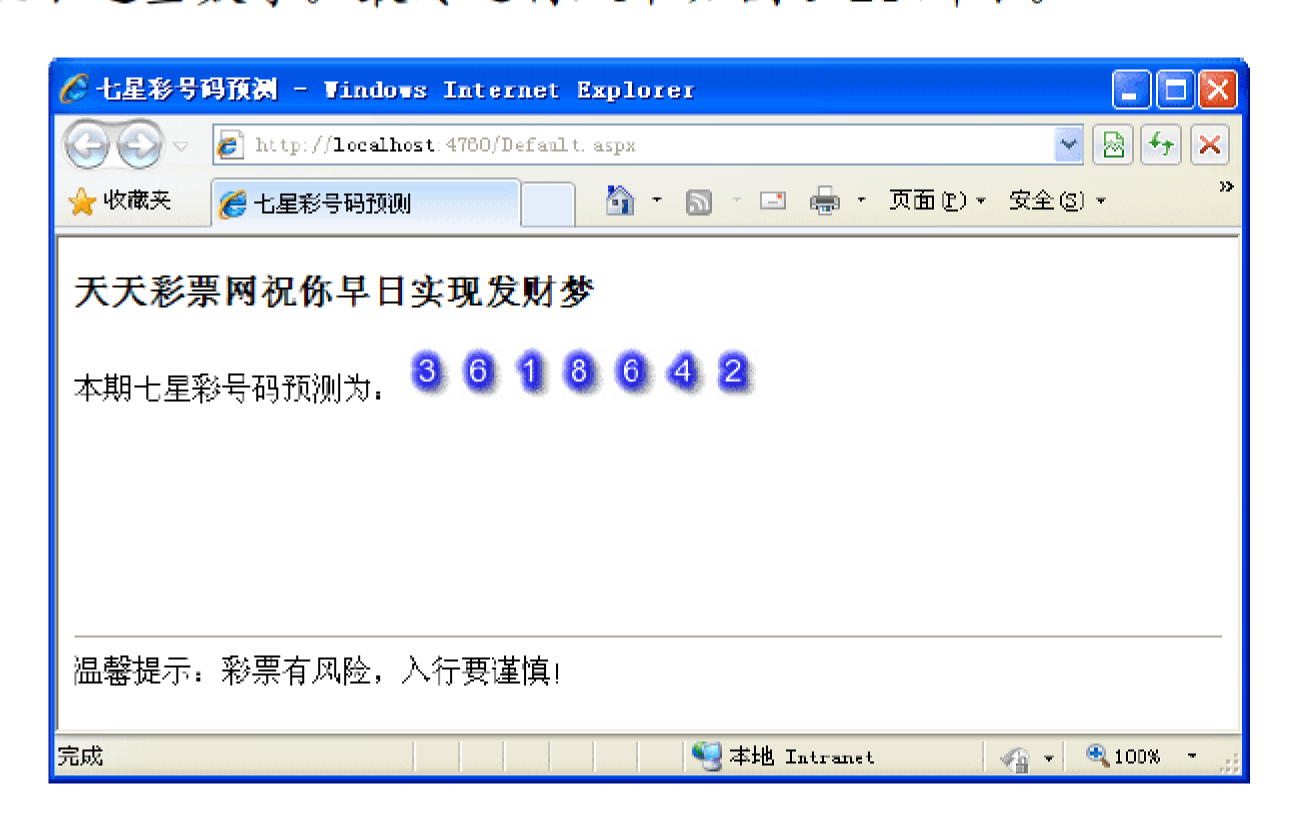
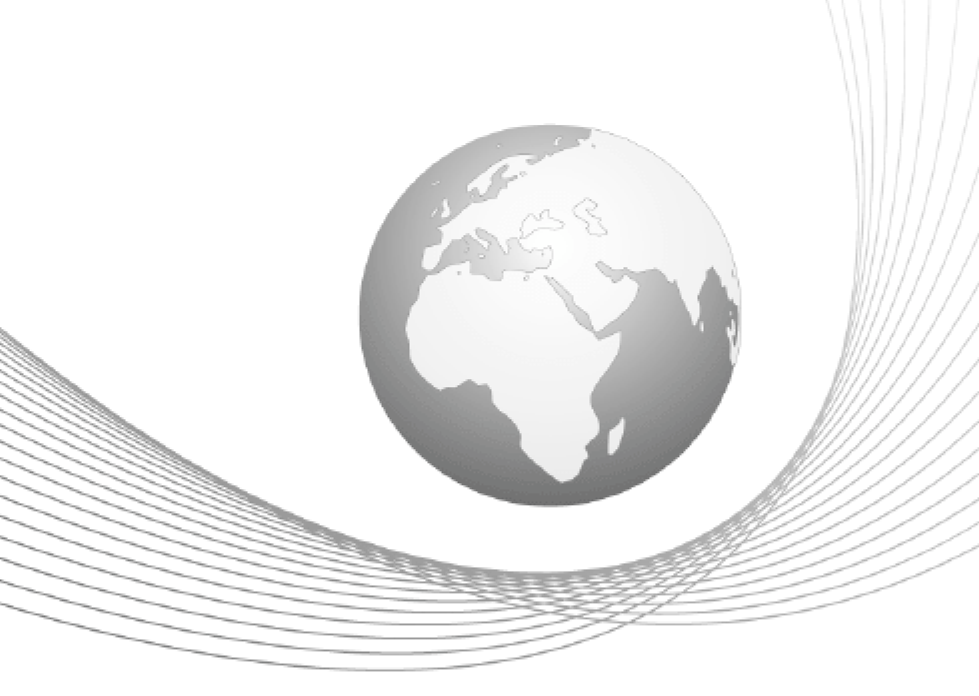


图 9-21　运行效果



第 10 章　安全性和验证

内容摘要：

Web 服务使用了非常灵活的开放性的标准，所以 Web 服务成为在网络中应用程序之间通信的一种绝佳的通信机制。我们可以通过它向客户端提供各种功能。但是，由于 Web 服务的标准的限制和 Web 服务需要支持各种类型的客户端，所以我们可以使用任意的方式来访问这些 Web 服务。所以，保护 Web 服务的安全是一件非常具有挑战性的事情。

由于安全性涉及诸多方面，而且 SOAP 规范中并没有提及安全性这一事实，所以有些人可能会认为安全性与 Web 服务无关。这种观点是不正确的。

当然，也不能低估了 ASP.NET 中的 Web 服务，因为我们可以采取许多措施来创建安全的 Web 服务。

本章将深入了解 ASP.NET 中关于 Web 服务的身份验证、授权机制，以及使用 SOAP 实现安全通信的技术，以保护 Web 服务的安全。

学习目标：

- 了解什么是安全机制
- 熟悉 ASP.NET Web 服务中的安全体系
- 熟悉 Windows 下的各种身份验证方式
- 熟练使用 Windows 身份验证机制
- 熟练使用 ASP.NET 提供的 Form 验证机制
- 熟练使用自定义 SOAP 报头来提高应用程序安全

10.1 了解 Web 服务安全机制

如何保护 Web 服务从来没有被官方着重强调过，但是 Web 服务肯定是要发布在 Internet 中的，肯定是要由用户来访问的。而我们不可能保证所有访问 Web 服务的用户都是遵规守矩的合法用户，所以 Web 服务可能会受到各种各样的攻击，比如 DDOS 攻击、非法用户试图进入、未授权的用户访问等。

不仅仅在 Internet 中的 Web 服务需要承担很大的风险，在内部网上发布的 Web 服务也同样可能会受到各种各样的外部攻击。

所以，我们有必要基于各个方面的考虑来设计 Web 服务。

 **视频教学：**光盘/videos/10/了解 Web 服务安全机制.avi　　长度：18 分钟

设计一个高安全性的 Web 服务是非常有必要的，所以我们不得不从各个方面来考虑 Web 服务的安全性。下面就来了解一下 Web 服务的安全机制。

10.1.1 基础知识——什么是安全机制

安全机制是在一个公共场合中保护合法权益不受侵犯的一些手段。

在应用程序中，我们可以通过保护私有属性和通过特定的身份验证来允许指定的用户合法地访问一些公有属性的机制以保证应用程序安全。这个身份可以是一组用户名和密码，也可以是一个密钥或一个证书等。

在 Internet 中使用安全机制来保护应用程序安全的时候，有一些名词需要了解，如下所示。

- 身份证：客户端用户的身份证明，可能是一组用户名和密码，也可能是一个证书或一个密钥等。
- 验证：验证是一种对客户端用户的权限进行检验的机制。它可以接收用户的身份证，并进行用户权限的检验。
- 权限：通常是一个确认的信息，标示着该用户在应用程序中的可操作范围。
- 授权：授权是服务器授予某用户对某种资源的访问权限的一种处理过程。
- 加密：加密是信息发送端和信息接收端协商制定的一种规则。我们可以使用它来进行数据转换，以确保数据的实际信息只能由信息发送端和信息接收端知道。

10.1.2 基础知识——Web 服务的安全体系

Web 服务运行在 Web 服务器中，而 Web 服务器需要有服务器系统的支持。所以在 Internet 中，对于整个 Web 服务的安全性设计需要考虑以下三个级别。

- 平台/传输级(点对点)安全性
- 应用程序级(自定义)安全性

● 消息级(端对端)安全性

当然这三种级别的安全性是针对 Web 服务的三个不同方向的设计，它们三个必须同时配合使用才能够保证 Web 服务应用程序是一个高安全性的应用程序。

1. 平台/传输级(点对点)安全性

平台/传输级安全性是指 Internet 中的两个平台之间的安全传输通道，它连接 Web 服务的客户端和服务器端，它在平台(如 Windows、Linux、UNIX 等)之间进行点对点的安全过滤。如图 10-1 所示为平台/传输级安全性的传输示意图。

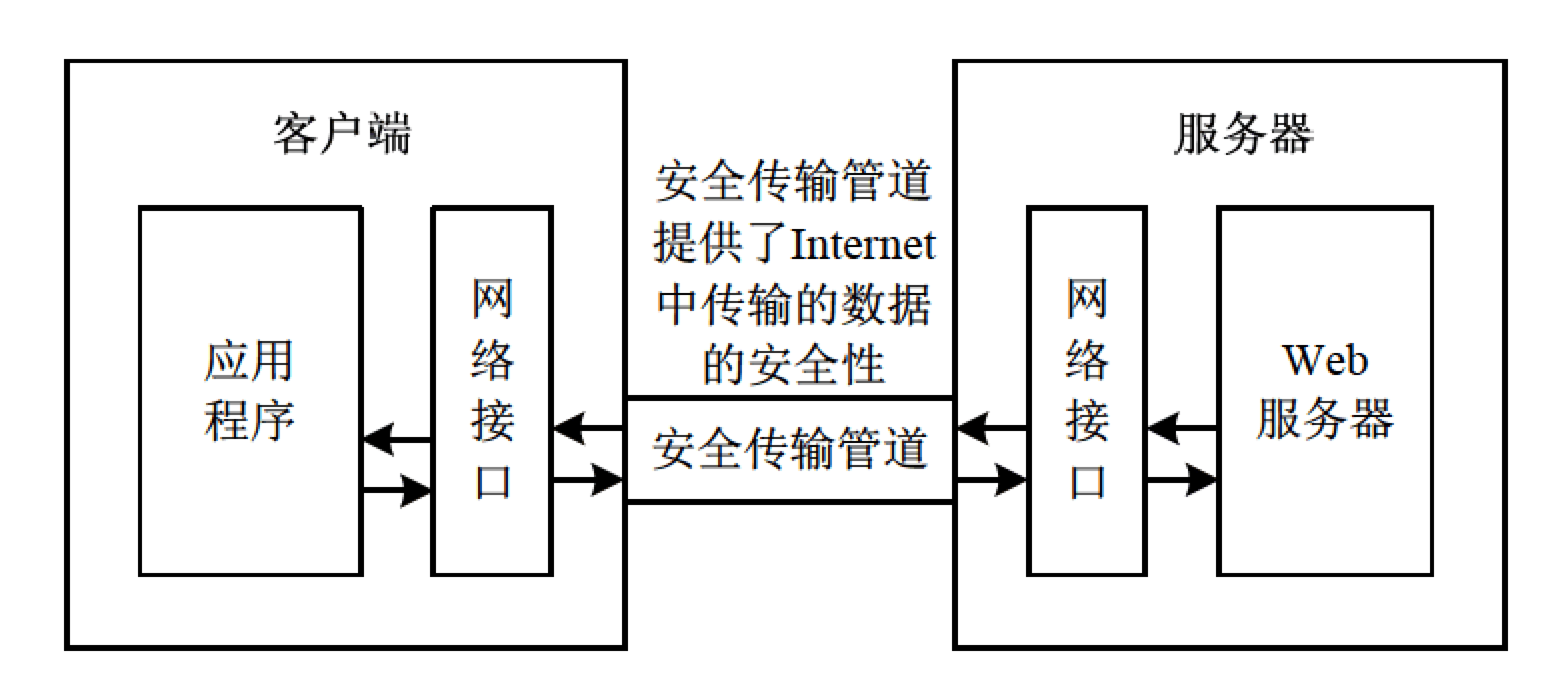


图 10-1　平台/传输级安全性的传输示意图

以 Windows 平台下的 ASP.NET 应用程序为例，该平台下的应用程序安全性可以设置下面三个方面的配置。

● Web 服务器 IIS 提供了最基本、集成和证书身份验证。
● ASP.NET Web 服务可以继承某些 ASP.NET 身份验证和授权功能。
● 还可以使用 SSL 或 IPSec 提供消息的完整性和机密性。

平台/传输级安全模型简单明了，除了 Web 服务，它还可以用于许多其他的应用程序解决方案。在这些应用程序解决方案中，平台/传输级安全模型都可以严格地控制传输机制和终结点的配置。

不过，这种方式的安全模型还有一些不太完美的地方。因为它基于平台的特性，所以它的安全性取决于基本的平台、网络传输机制和安全性服务提供程序，并与它们紧密集成。

2. 应用程序级(自定义)安全性

应用程序级安全性是指由应用程序负责提供的安全性机制，所有的安全性控制全部由应用程序自己定义。

例如可以在 Web 服务应用程序中使用自定义的 SOAP 消息头传递用户凭证，以便根据每个 Web 服务请求对用户进行身份验证。

3. 消息级(端对端)安全性

消息级安全性是指由网络中发送的消息本身来控制数据的安全性。

消息级安全性是一种使用非常灵活而且功能非常强大的方法。GXA 提案使用的就是这种方法。

消息级安全性的示意图如图 10-2 所示。

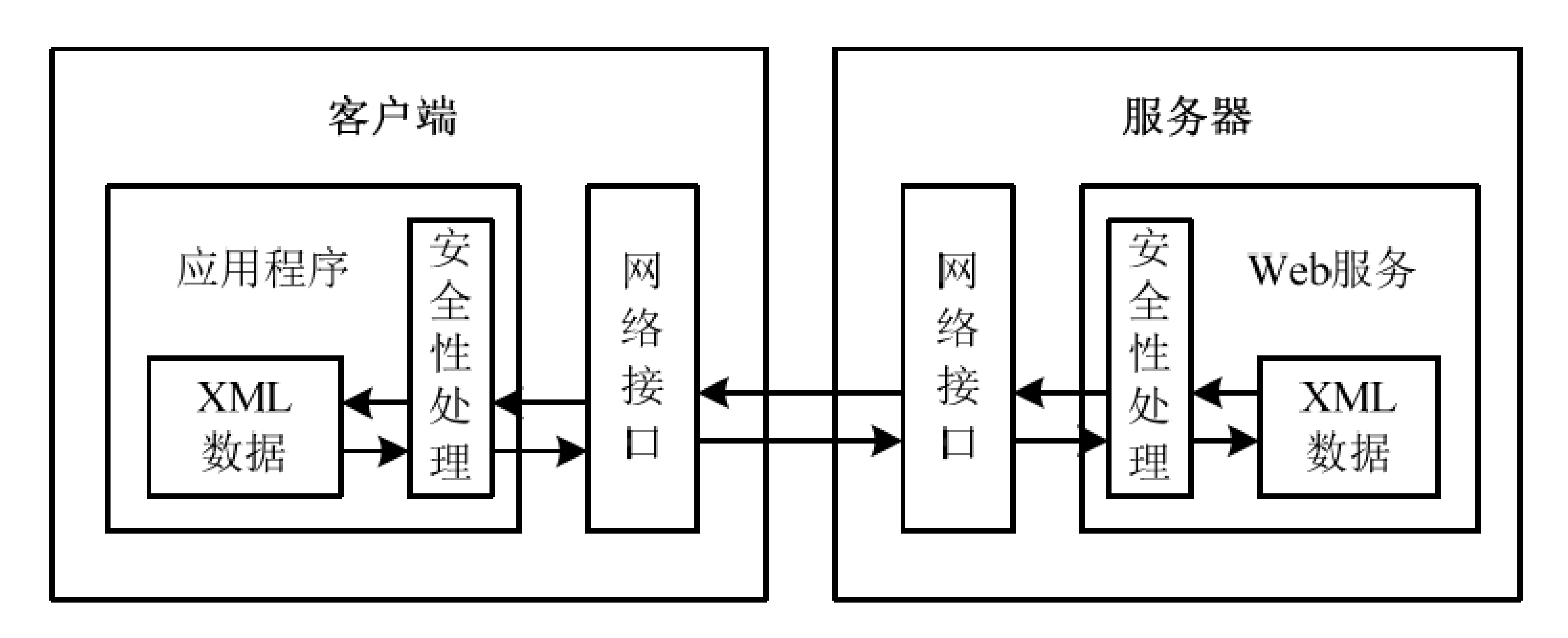


图 10-2　消息级安全性示意图

技术文档	GXA

GXA：全球 XML 体系结构。

可以将 GXA 定义为一种框架，一系列的协议，一个规范集，一个视窗或是一种标准的起始。但是事实上，GXA 是上述方面的综合。

定义 GXA 的一个最简单的方法就是将 W3C 创建的规范统一封装起来。当前所有的 WS-I 规范都来自 GXA。

GXA 协议集是建立在一系列的共享设计原则之上的。这些设计原则为所有新生的或是已存在的 GXA 协议起着指导方针的作用。

- 分散和联盟：由于 XML Web 服务在应用程序的消息机制中没有集中管理功能，因此它是分散的。而在联盟中 Web 服务使用命名空间和 URI 机制保证层次结构的统一。
- 模块性：GXA 框架协议能利用 GXA 框架的构造基础：SOAP。针对具体问题，GXA 能分解为许多构建模块有针对性地处理每个问题。
- 以 XML 为数据模型的基础：GXA 采用广泛使用的 XML1.0 标准作为数据模型的基础。
- 传输中立：GXA 在 SOAP 信息交换层被完全规范，并不依赖于用于通信的传输层的语法和语义。
- 应用领域中立：GXA 协议表达的是一种规范，这种规范用于解决跨越许多应用领域的广泛的问题。由于 GXA 协议并没有涉及特殊的应用需求，我们就可以用它们作为固定网络服务器框架的基准，同时允许应用的一些具体细节。

GXA 的魅力所在就是它为网络服务提供了一个在精确定义以及安全模式下进行操作的基础，同时是特殊应用设计的负载并完全独立于这些协议本身。

SOAP 消息传递的增强功能提供了消息完整性、消息机密性以及单次消息身份验证等校验机制。我们可以使用在 SOAP 报头中传递的安全令牌提供客户端用户的身份验证功能；也可以使用数字签名来提供安全通信的功能，以确保消息的完整性；还可以使用 XML 加密来保证 SOAP 消息的机密性。

总体来说，消息级安全性主要有以下优点。

- 可以不依赖于基本传输。
- 支持异类安全体系结构。
- 提供端对端的安全性并通过中间应用程序节点提供消息路由。
- 支持多种加密技术。
- 支持不可否认性。

10.1.3　基础知识——与 Web 服务有关的安全选项

在 ASP.NET 应用程序中，Web 服务支持多种类型的安全设置选项，分别是：

- SSL
- 简要验证
- 集成 Windows 验证
- 客户授权证书验证
- 自定义验证

基本验证、简要验证和集成 Windows 验证方法作为一项服务由 IIS 提供给 Web 服务。ASP.NET 提供了表单、护照验证和 SOAP 技术，ASP.NET 和 Web 服务通过使用验证提供者来处理这种验证。

10.1.4　Web 服务安全层

如果要保护一个 Web 服务，我们可以从三个层次着手，如表 10-1 所示。

表 10-1　Web 服务的安全层

层	安全类型	说　明
应用层	HTTP 身份验证、密码技术	程序开发人员可以在这里使用代码进行一些验证
TCP 层	SSL	可以进行一些少量的编程工作
IP 层	IP 安全	该层对编程人员是透明的，但只限于 Windows 平台和 UNIX 平台

下面从低到高依次了解这些安全层。

1. IP 层安全

IP 安全是一种 Internet 标准，用于保护在 IP 层传输的 Internet 数据流量。IP 安全在 IP 层操作，因此对编程人员来说是透明的。

可以通过向 IP 包添加两个题头来操作该协议，其中一个题头用于身份验证，另一个用于数据加密。

身份验证机制是为了保证数据的接收者确信信息的发送者是一个真实合法的信息发送者。而且，加密技术可以确保数据的机密性，其方法是以预定的方式加密编码被保护的数据。

IP 安全机制允许几乎所有的安全机制完成身份验证和加密工作。但是这可能会产生各平台

间因为相互竞争而导致的不兼容。所以，基于这种模型的安全机制比较适合于基于 Web 应用程序的 LAN 和 WAN，所有参加通信的机器都被一个具有统一标准的系统所控制。

不过，IP 安全的另一个局限性是它没有为编程人员提供一个标准的方法，以使他们确保自己的应用程序仅能运行在基于 IP 安全的系统之上。也就是说，IP 层可以很简单地被不法用户绕过。

2．TCP 层安全

TCP 层安全通常以使用 SSL(Secure Sockets Layer，安全套接层)而知名。SSL 可以为 HTTP 提供透明的加密机制。

TCP 层安全是一个被广泛依赖的安全机制，我们可以使用它来确保基于 HTTP 的通信的完整性。

因为 ASP.NET Web 服务是基于 HTTP 之上的 SOAP 协议的，所以它也必须使用 SSL 来保护数据的完整性。

3．应用层安全

.NET Framework 也提供了一种基于 HTTP 身份验证的机制。HTTP 身份验证的机制是一种基于用户名和密码的身份验证机制。也就是说可以设置在使用 HTTP 协议访问应用程序的时候需要用户使用用户名和密码来访问这个 Web 服务器。



虽然 HTTP 可以设置需要用户名和密码来访问该服务器，但是一般来说，Web 服务器都会开启匿名登录的功能，所以一般访问 Web 服务器的时候不需要输入用户名和密码。

在前面我们说过，为了自身加密的需要，HTTP 层也需要依赖于 SSL。

要保护 Web 数据不被拦截窃取，可以编写安全通道收发器，用来处理数据的安全问题。另外保护 Web 服务数据完整性的另外一种方法是使用加密技术，这样可以使非法用户即使得到数据也不能知道数据的实际内容。

10.2　使用 Windows 验证来限制用户访问

在 Web 出现的早期，我们对网络中用户访问的控制都是由系统账户控制和管理。随着 Web 功能的日益强大，用户权限验证方式也多种多样，但是使用 Windows 验证仍然是一个不可或缺的方案。



视频教学：光盘/videos/10/Windows 验证.avi　　长度：13 分钟

10.2.1　基础知识——集成 Windows 验证

当请求一个使用集成 Windows 验证的 Web 站点时，Web 服务器域名和一个标记将返回到服务器(这个标记包括使用散列算法加密过的密码和用户名等信息)。接着 Web 服务器发送用户名、域名等信息给域名控制器，域名控制器将验证身份的合法性并给 Web 服务器(如 IIS)返回

一个响应信息。

1．集成 Windows 验证的优点

前面也提到过，Web 应用程序可以使用多种身份验证方法。其中使用集成 Windows 验证有诸多优点，如下所示。

- 容易实现：Windows 验证基于 Windows 系统的安全机制，使用起来不需要太多额外的技术。
- 能完美地被 IIS 支持：IIS 和 Windows 系统同出一家，当然在结合性上更加紧密。使用 IIS 实现 Windows 验证的功能非常简单。
- 无过高要求：配置起来非常简单，我们只需要在 Windows 视窗界面中单击几下鼠标即可完成配置。
- 密码从不通过网络发送：Windows 验证会在客户端将用户密码使用散列算法进行加密，因此保证了密码的安全性。

2．集成 Windows 验证的缺点

任何事物都有利有弊，技术也一样。集成 Windows 验证主要有以下几个缺点。

- 浏览器兼容性不是很好：它可以被 IE 等最为主流的浏览器所支持，但是其他的一些浏览器对 Windows 验证的支持还不是很好。
- 集成 Windows 验证不能应用在代理服务器上。
- 可能面临服务器端口被占用的问题：附加的 TCP 端口可能会被防火墙拦下，所以我们要使用集成 Windows 验证，还需要在防火墙上打开指定的附加端口。
- 不能靠自定义权限使用此验证：在我们使用基本验证时，身份证将一直与 Windows 账号数据库相对比。我们不可能以自定义权限来比较身份证，因为它使用了 Windows 用户账号来做这些事情。

3．何时使用 Windows 验证

集成 Windows 验证是微软对 HTTP 协议的扩展。它作为特有的功能被添加到 IIS 服务器和 IE 浏览器中。

因为 Windows 验证对浏览器的支持不是很好，所以在 Web 应用程序中，集成 Windows 验证通常被应用在企业内部网中，也好进行客户端统一。

由于 Windows 验证是发生在客户端的，也就是说基本上 Windows 验证和服务器端没有关系，所以在 Web 服务器中使用集成 Windows 验证的方式比较安全。但是，在服务器端必须同时为该验证打开附加的 TCP 端口，而打开额外的端口可能会给企业内部网络带来安全隐患，很多网络管理员并不喜欢这么做。

所以，我们在是否使用 Windows 验证的问题上要仔细考虑以便进行合理的设计。

10.2.2　实例描述

在 Windows 服务器系统中，IIS 对 Windows 验证方式提供了很好的支持，我们可以在 IIS

中很轻松地对 Web 站点启用 Windows 验证模式。

本实例，我们就在 Windows 2003 系统的 IIS 服务器中启动 Windows 验证，对 Web 服务进行第一道的安全验证。

10.2.3 实例应用

【例 10-1】 使用 Windows 验证来限制用户访问

(1) 使用 Visual Studio 2010 开发工具创建一个 Web 项目，并在其中创建一个 Web 服务，命名为 WSCounter。

(2) 在 WSCounter 中创建一个名为 Add 的 Web 方法，执行加法操作。WSCounter 类的代码如下所示：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
namespace WebServices
{
    /// <summary>
    /// WSCounter 的摘要说明
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // 若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
    // [System.Web.Script.Services.ScriptService]
    public class WSCounter : System.Web.Services.WebService
    {
        [WebMethod]
        public decimal Add(decimal a, decimal b)
        {
            return a + b;
        }
    }
}
```

(3) 此时由于安全级别的限制，Web 服务方法还不能被远程计算机的浏览器访问测试。下面修改 web.config 文件，添加相应的配置代码。

这里需要在 configuration 节点下的 system.web 节点中添加一段配置信息，如下所示：

```
<configuration>
    <!—其他配置省略 -->
    <system.web>
        <!—其他配置省略 -->
        <webServices>
            <protocols>
                <add name="HttpGet"/>
```

```
            <add name="HttpPost"/>
          </protocols>
        </webServices>
    </system.web>
</configuration>
```

(4) 生成该 Web 项目，并将其发布成一个 Web 站点。

(5) 将该 Web 站点上传到 Web 服务器默认站点的根目录中。这里 Web 服务器使用的是 Windows 2003 系统和 IIS 6.0 的 Web 服务器程序。

(6) 接下来需要在 IIS 中启用 Windows 验证方式。首先需要依次展开 Internet 信息服务左侧窗口中的【本地计算机】节点下的【网站】节点，然后右键单击【默认网站】节点，在弹出的快捷菜单中选择【属性】命令，打开【默认网站 属性】对话框。我们需要设置该网站的安全性，所以切换到【目录安全性】选项卡，如图 10-3 所示。

在【目录安全性】选项卡中可以看到 IIS 提供了三大类的安全性验证方法，其中包括身份验证和访问控制、IP 地址和域名限制、安全通信。

(7) 在【目录安全性】选项卡中单击【身份验证和访问控制】选项组中的【编辑】按钮，打开【身份验证方法】对话框，如图 10-4 所示。

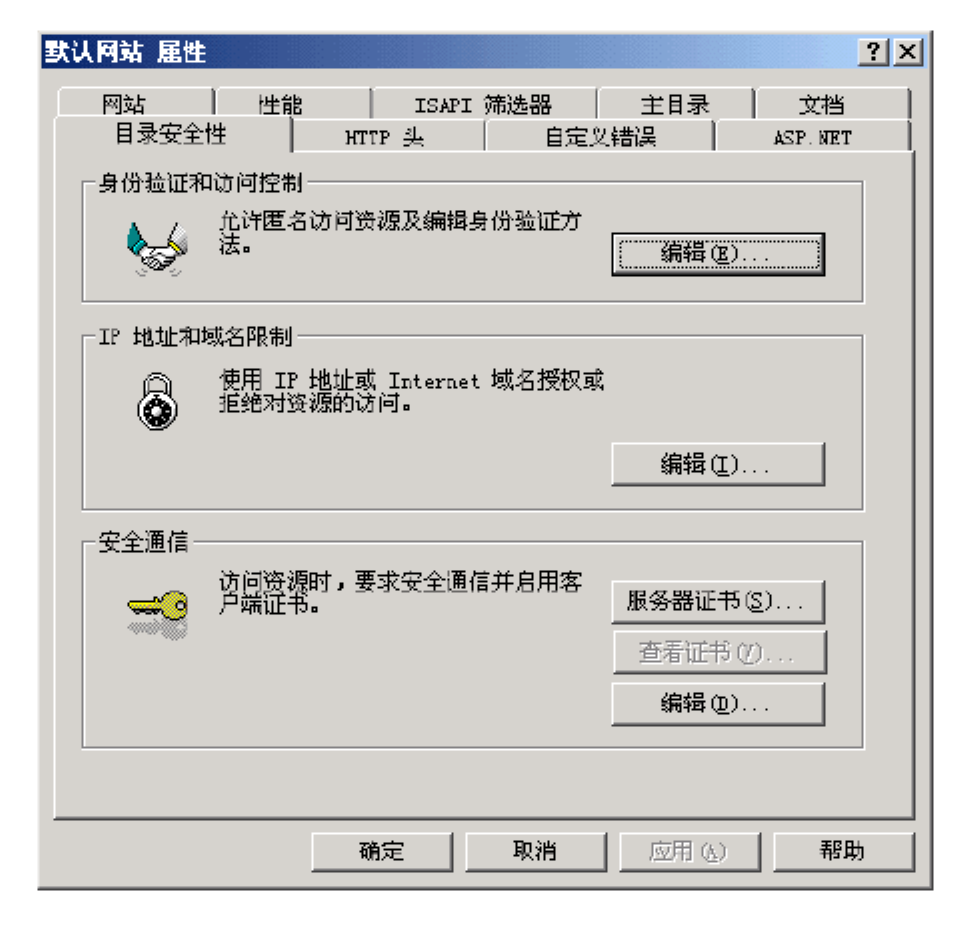


图 10-3 【目录安全性】选项卡

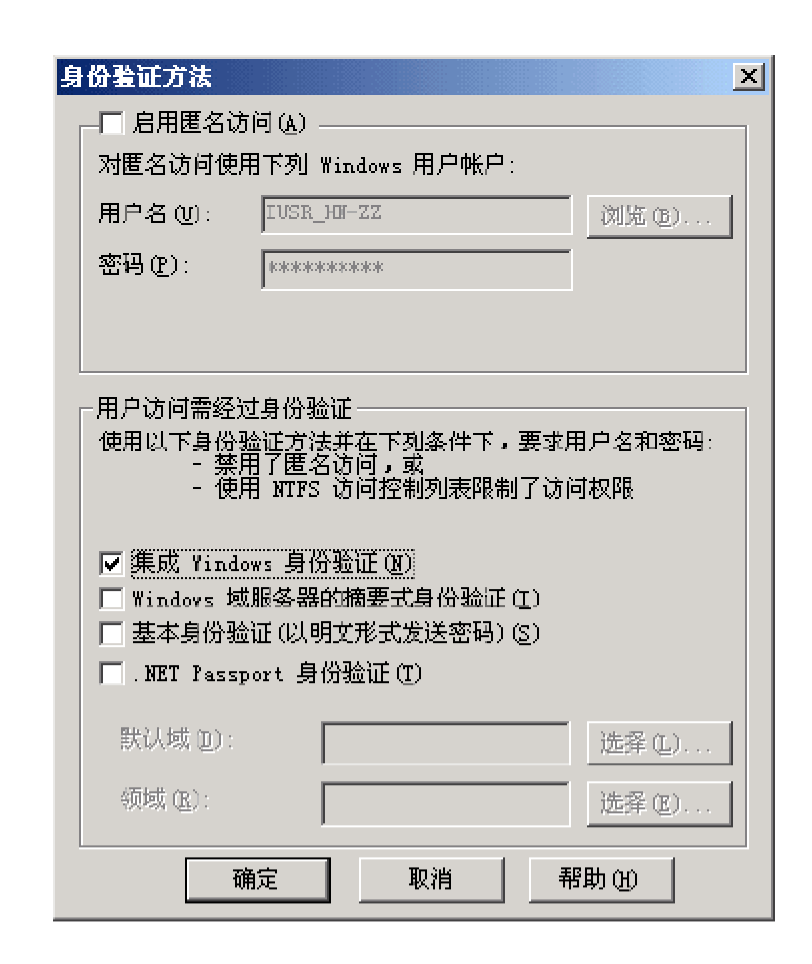


图 10-4 【身份验证方法】对话框

(8) 在【身份验证方法】对话框中取消对所有复选框的选择，然后选中【集成 Windows 身份验证】复选框，就启用了对该 Web 站点的 Windows 验证模式。

(9) 单击【确定】按钮保存设置。

10.2.4 运行结果

前面已经将 Web 服务发布到 Windows 2003 系统的 IIS 中，我们可以直接在客户端使用浏览器对该 Web 服务进行测试。

打开 Internet Explorer 8 浏览器，使用服务器 IP 访问 Web 服务(比如这里的路径为 http://192.168.0.12/WSCounter.asmx)。访问该路径以后，系统就会自动弹出一个【连接到

192.168.0.12】对话框，如图 10-5 所示。

在这里需要输入服务器系统中的一组用户名和密码，如果输入错误，将会要求重新输入用户名和密码，如图 10-6 所示。



图 10-5 【连接到 192.168.0.12】对话框

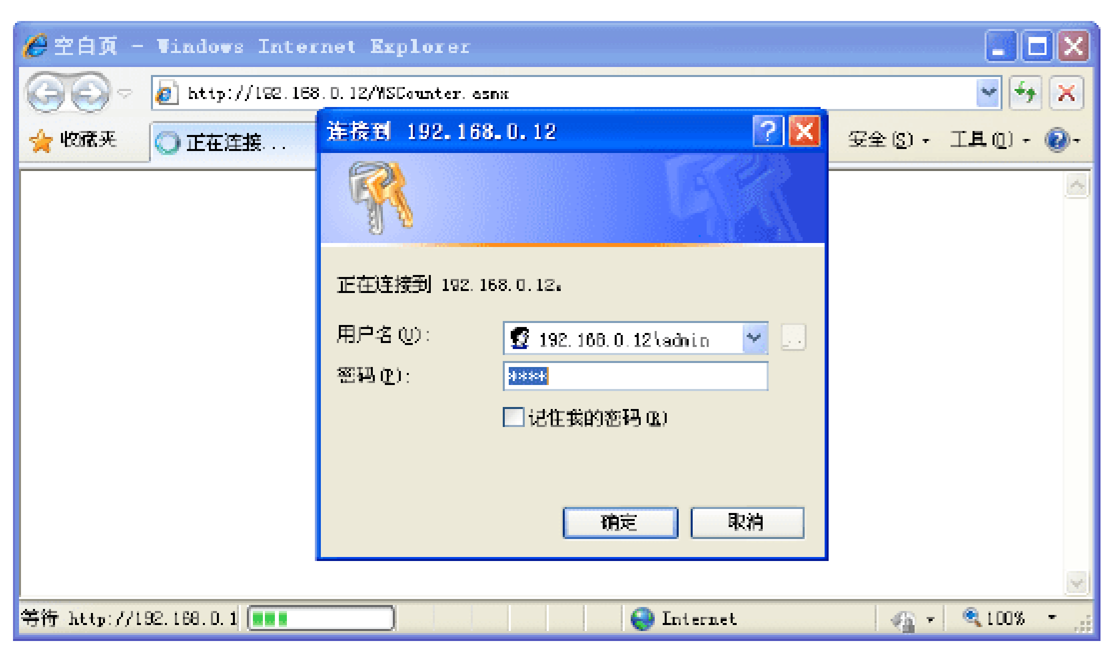


图 10-6 重新登录

当输入一组在服务器中存在的合法的用户名时，系统就会登录成功，打开相应的页面，如图 10-7 所示。

接下来就可以测试该 Web 服务方法了，如图 10-8 和图 10-9 所示。

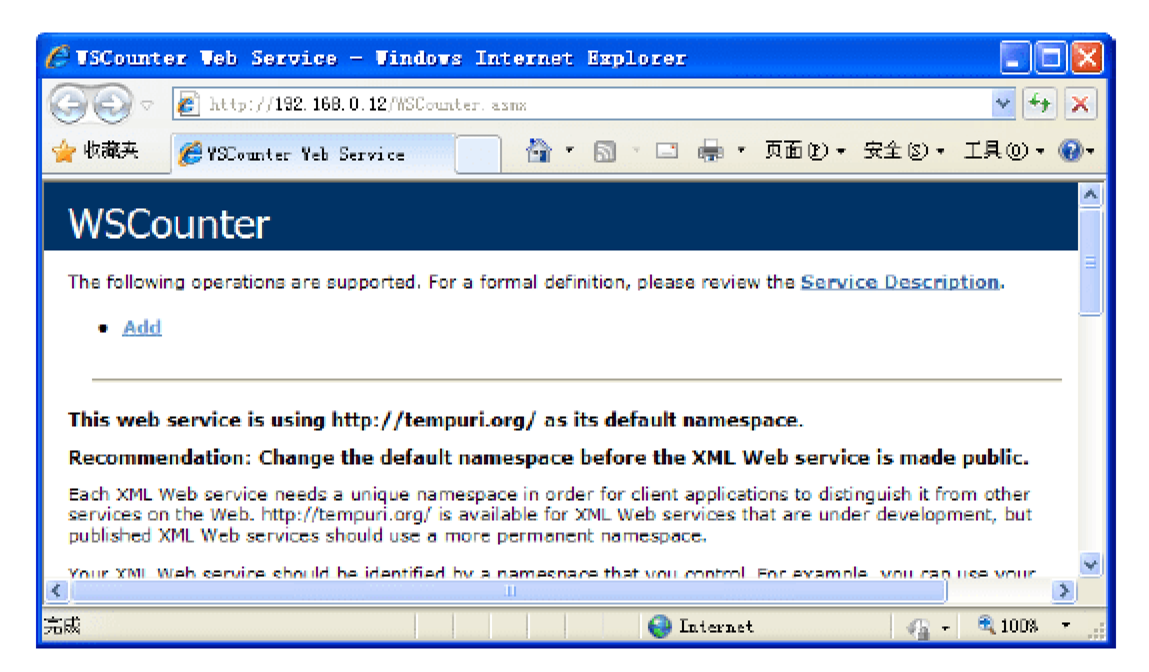


图 10-7 登录成功

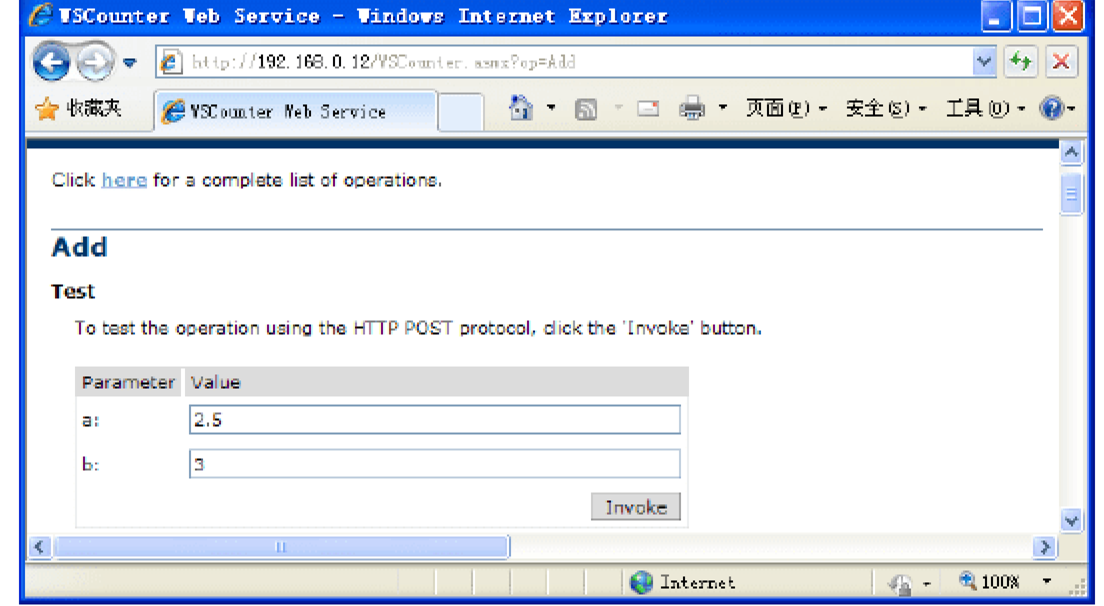


图 10-8 Web 服务方法 Add



如果没有在第 3 步对 web.config 文件的配置，这里测试的时候就不会显示输入参数的表单，而是一个“The test form is only available for requests from the local machine.”的提示。

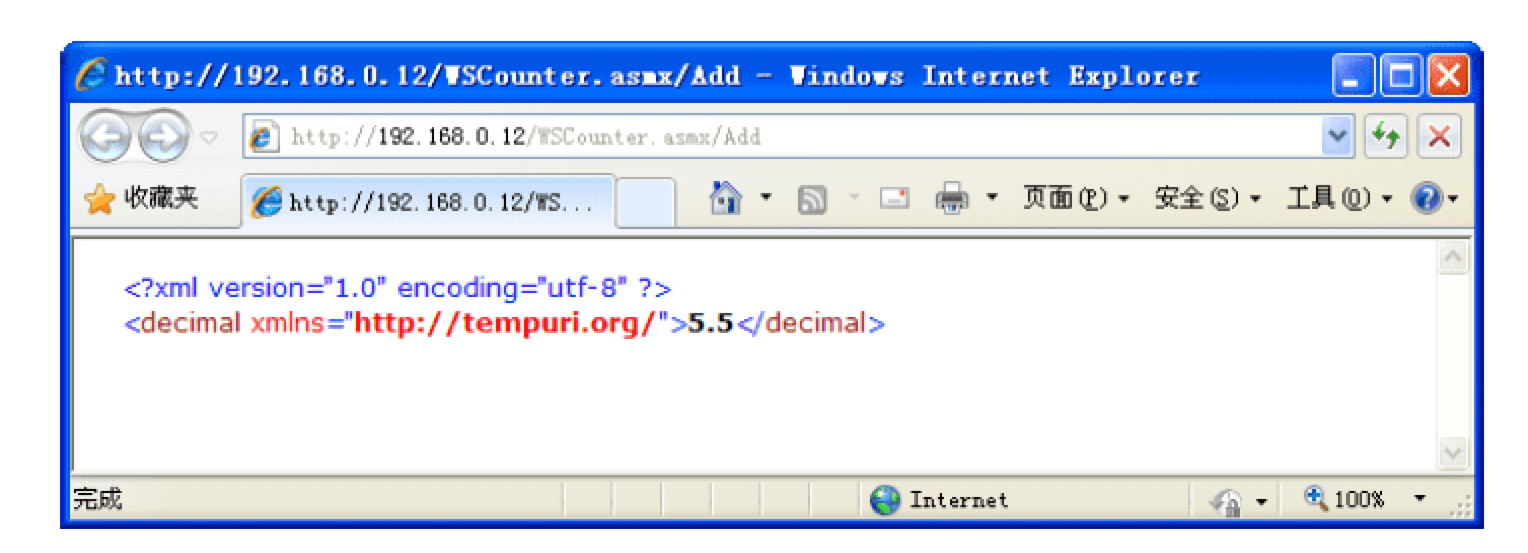


图 10-9 Web 服务方法 Add 执行结果

10.2.5　实例分析



源码解析：

本实例，首先使用 VS2010 创建一个 Web 项目，并添加一个用于执行数学运算操作的 Web 服务，然后在该 Web 服务中添加相应的 Web 服务方法。将该 Web 项目发布到 Web 服务器上，配置 IIS 的安全验证方法为“集成 Windows 身份验证”即可启用该 Web 站点的 Windows 身份验证功能。这时，再来访问 Web 服务时就需要使用 Windows 服务器中存在的合法用户了。

10.3　使用表单验证进行权限过滤

当 Web 服务有非常多的用户量，而且需要关心 Web 服务系统的安全性和可伸缩性时，使用表单验证来实现用户身份认证将是一个不错的选择。

表单验证功能是完全使用自己定义的数据库和验证方法来进行的用户权限验证。由于我们使用的是完全自定义的方法，所以可以自由地控制验证过程中的细节问题。



视频教学：光盘/videos/10/表单验证.avi　　　　长度：5 分钟

表单验证允许采用自定义权限的方法来验证用户。表单验证是完全自定义的，并且提供了内置功能来管理 Cookie，全权负责有关加密、解密、验证的复杂事项。当用户未被授权或验证失败时，可以把用户转送到注册界面或登录界面中。

当请求一个使用表单验证的 Web 站点时，Web 服务器会把请求发送到 ASP.NET 运行时组件，ASP.NET 运行时组件将会检查 Web 请求是否包含一个与服务器端相对应的 Cookie 信息。如果这个 Cookie 在 Web 请求里存在，那么 ASP.NET 运行时组件将会把请求传送到 Web 服务页面。如果 Cookie 不存在，那么 ASP.NET 运行时组件将把 Web 请求发送到 web.config 文件中定义的注册页面。

表单验证是当今站点使用的非常流行的验证方法，表单验证提高了以各种权限确认身份的灵活性。基于各种需求，可以有选择地定制一些验证方法来实现对用户身份的验证。

1．表单验证的优点

使用表单验证的方式实现 Web 服务的安全性验证有诸多优点，如下所示。

- 易于实现：只需要简单配置 web.config 文件，并使用数据库或活动目录建立一个验证机制即可。
- 无特殊要求：ASP.NET 运行时组件可以自动管理 Cookie。
- 支持自定义权限：可以自定义一些权限来实现对不同用户的验证。
- 灵活性强：因为表单验证完全支持自定义的权限验证，所以这给我们程序开发人员提供了极大的灵活性，使我们更容易控制整个验证过程。

2．表单验证的不足

当然，使用表单验证来实现 Web 服务的安全性验证机制也不是最完美的，总体来说还有以下不足。

- 需要 Cookie：表单验证使用 Cookie 验证用户，如果客户端不支持 Cookie，那么验证机制将会失效。
- 需要耗费更多的服务器性能：表单验证要求更多的内存和更强的处理能力来满足服务器的性能需求。因为它要处理外部权限，所以对系统的性能有较高的要求。

10.4　禁止使用浏览器访问 Web 服务

Web 服务是发布在 Internet 上的。默认情况下，它可以被客户端以 HTTP 方式访问。

访问一个 Web 服务可以使用 POST、GET 和 SOAP 三种方式。一般来说，使用编程方式请求 Web 服务时都是使用 SOAP 方式，而我们使用 Web 浏览器请求 Web 服务时是 GET 方式或 POST 方式。

某些时候，由于一些比较保密的 Web 服务并不想让用户使用浏览器访问，那么，就可以在服务器端设置禁用浏览器访问。

**视频教学**：光盘/videos/10/禁止浏览器访问.avi　　长度：16 分钟

10.4.1　基础知识——禁用 GET、POST 请求

默认情况下，Web 服务客户端可以使用以下三个方式与 ASP.NET Web 服务进行通信：HTTP GET、HTTP POST 和 HTTP SOAP。

如果 Web 服务不需要客户端使用浏览器进行访问，那么可以禁止客户端使用 HTTP 协议的 GET 方式和 POST 方式访问 Web 服务。

Web 服务正常运行的情况下，Web 客户端不需要使用 GET 和 POST 这两种方式。如果禁用这两个协议，可以很大程度上避免出现潜在的安全隐患，使 Web 页面无法访问在防火墙后面运行的 Web 服务。

当然，禁用这些请求方式意味着，客户端将无法使用 Web 服务测试页中的“调用(Invoke)”按钮来测试 Web 服务。所以，必须开发一个简单的应用程序并添加对该 Web 服务的引用，以执行对该 Web 服务的测试工作。



无论在 Web 服务中禁用或启用任何一种访问方式，都不会对本地访问构成任何影响。

如果要在整个服务器中禁用 Web 服务的 GET 和 POST 请求方式，可以在.NET Framework 中的 machine.config 文件中配置 webServices 元素节点，示例代码如下所示：

```
<webServices>
    <protocols>
```

```
    <add name="HttpSoap"/>
      <!-- <add name="HttpPost"/> -->
      <!-- <add name="HttpGet"/>  -->
    <add name="Documentation"/>
   </protocols>
</webServices>
```

这里配置了客户端可以使用 HTTP SOAP 方式请求 Web 服务，以及可以访问 Web 服务的文档页。



webServices 节点在配置文件的 configuration 节点下的 system.web 节点中。

从上面代码中可以看到 webServices 元素中的 protocols 元素中可以使用 add 元素配置 Web 服务请求的访问方式，其 name 属性可以设置请求的方式，可选值如表 10-2 所示。

表 10-2　add 元素的 name 属性可选值

值	说　明
HttpGet	添加 HTTP GET 方式。传递的参数被追加到 HTTP 请求 URL 的查询字符串中。返回值被当作简单的 XML 文档放入 HTTP 响应的正文中
HttpPost	添加 HTTP POST 方式。传递的方法参数被放在 HTTP 请求的正文中。返回值被当做简单的 XML 文档放入 HTTP 响应的正文中
HttpSoap	添加 HTTP SOAP 方式。SOAP 消息在 HTTP 请求的正文中发送。SOAP 响应在 HTTP 响应的正文中发送
Documentation	添加特殊的 Documentation 方式。当在启用了此方式的情况下直接请求.asmx 页时，ASP.NET 运行 Helper 页以创建 HTML 文档页，该文档页被传递到提出请求的客户端

当然，可以在 protocols 元素中使用 add 方法添加服务器允许接受的请求方式，也可以使用一些其他方法移除或清除前面的配置。protocols 元素可以接受的 3 个子元素及其说明如表 10-3 所示。

表 10-3　protocols 元素的子元素

名　称	说　明
add	添加 ASP.NET Web 服务可用来接收从客户端发送的请求数据和返回响应数据的指定协议。默认情况下，仅启用 HttpSoap 和 Documentation
clear	从配置文件的范围内移除所有的协议
remove	从配置文件的范围内移除用来处理请求和响应数据的指定协议

关于这三个配置节点，示例代码如下所示：

```
<webServices>
   <protocols>
      <add name="protocolname"/>
```

```
        <remove name="protocolname"/>
        <clear>
    </protocols>
</webServices>
```



如果要配置整个服务器的配置选项，可以配置.NET Framework 的 machine.config 文件。而如果只配置 Web 站点的配置选项，可以在该站点的 web.config 文件中配置。

10.4.2 实例描述

禁用 GET、POST 请求可以阻止用户使用浏览器访问 Web 服务。这样在一定程度上提升了 Web 服务的安全性。

下面我们就以例 10-1 中用过的实例来演示禁用 GET、POST 请求的方法。

10.4.3 实例应用

【例 10-2】 禁止使用浏览器访问 Web 服务

(1) 先来修改 Web 服务项目的配置文件，即修改 webServices 节点下的 protocols 节点中的子节点，只添加 HttpSoap 和 Documentation 两项，而移除其余的两项，代码如下：

```
<configuration>
    <!—其他配置省略 -->
    <system.web>
        <!—其他配置省略 -->
        <webServices>
            <protocols>
                <add name="HttpSoap"/>
                <remove name="HttpGet"/>
                <remove name="HttpPost"/>
                <add name="Documentation" />
            </protocols>
        </webServices>
    </system.web>
</configuration>
```

(2) 重新将该项目上传到服务器上的 Web 站点中。

(3) 重新将服务器中的 Web 站点修改为默认的“启用匿名访问”，如图 10-10 所示。

(4) 现在需要一个 Windows 窗体作为客户端来调用这个 Web 服务。我们先创建一个 WinForm 项目，在该项目中添加到 Web 服务 http://192.168.0.12/WSCounter.asmx 的服务引用，命名空间名称为 ServiceCounter。

(5) 创建一个 Windows 窗体项，用于实现加法计算功能。窗体的界面如图 10-11 所示。

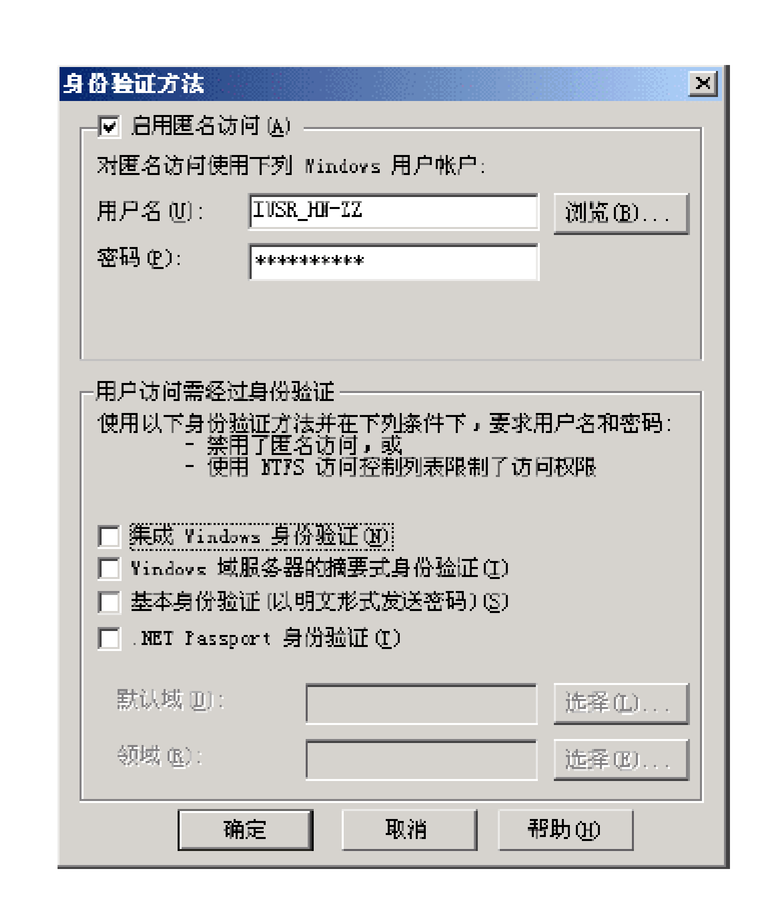




图 10-10　启用匿名访问

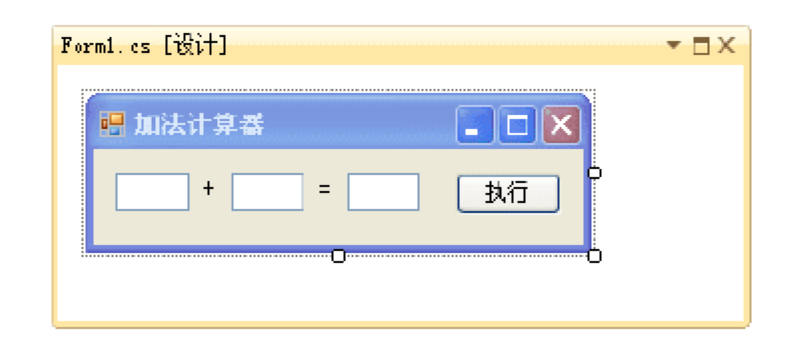


图 10-11　窗体设计效果

(6) 需要在单击【执行】按钮时，调用 Web 服务执行加法操作，所以这里在【执行】按钮的单击事件处理程序里添加执行代码，如下所示：

```
private void btnRun Click(object sender, EventArgs e)
{
    //创建代理类对象
    ServiceCounter.WSCounterSoapClient wsc =
        new ServiceCounter.WSCounterSoapClient();
    decimal n1 = decimal.Parse(this.txtN1.Text);
    decimal n2 = decimal.Parse(this.txtN2.Text);
    //请求 Web 方法，执行加法运算
    decimal result = wsc.Add(n1, n2);
    this.txtValue.Text = result.ToString();
}
```

10.4.4　运行结果

使用浏览器访问 Web 服务 http://192.168.0.12/WSCounter.asmx，打开 Web 服务的文档页面。接下来进入 Web 方法 Add 的页面，运行结果如图 10-12 所示。

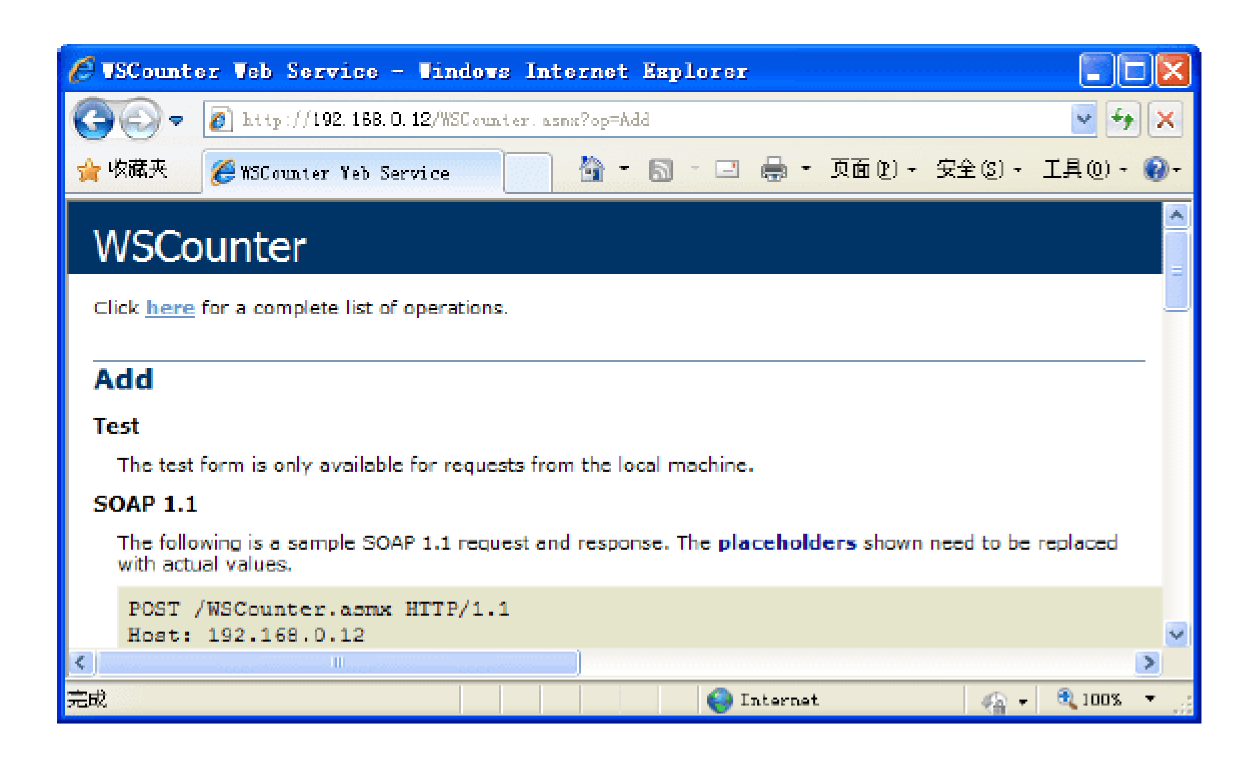


图 10-12　Web 服务方法 Add 的说明页面

从图中可以看到，Web 服务方法 Add 中没有测试的表单了。而是出现一行提示信息“The test form is only available for requests from the local machine.”，意思是“测试表单只对服务器本机有效”，也就是说在远端计算机中无法使用 GET 请求或者 POST 请求来测试 Web 服务。

下面再使用编写的测试程序来验证这个 Web 方法。

运行窗体应用程序，分别在前两个文本框中输入一个数值，然后单击【执行】按钮，稍等加法运算的结果就会被显示到第三个文本框中，如图 10-13 所示。

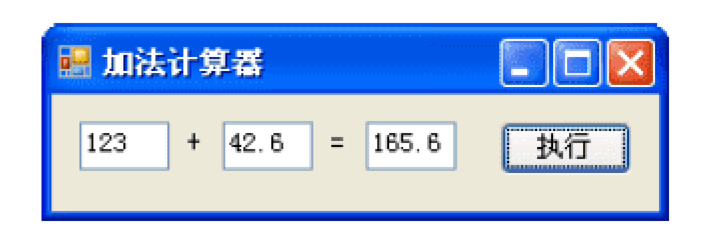


图 10-13　使用测试程序测试 Web 服务

10.4.5　实例分析



源码解析：

本实例，首先修改 Web 服务项目的配置文件，设置该项目中的 Web 服务可以在浏览器中查看文档，也可以使用 SOAP 方式请求，但是不能使用 POST 或 GET 方式请求。然后还要将服务器中的身份验证方式改为允许匿名登录。最后编写一个客户测试程序使用 SOAP 方式测试 Web 服务。

10.5　实现自定义验证的 Web 服务

在前面讲过，可以在 SOAP 消息头中保存少量的信息，比如用户的身份验证信息等。

在 Web 服务中，使用 SoapHeader 方法可以在 Web 服务方法的请求中增加头部信息，当有人调用 Web 服务时，可以通过查询这个请求的头部信息来获取该访客的身份，以实现系统的身份认证功能。



视频教学：光盘/videos/10/自定义验证.avi　　　　长度：13 分钟

10.5.1　基础知识——自定义 SOAP 报头

为了使应用程序能达到一定的目的，可以往 SOAP 报头中添加一些自定义的信息。比如可以向 SOAP 报头中添加指定的用户名和密码变量，然后就可以在 Web 服务中读取该 SOAP 报头信息，做出一些相应的验证。

SOAP 报头提供了一种方法，用于将数据传送到 Web 服务方法中，或从 Web 服务方法中传递数据，并且该数据不直接与 Web 服务方法的主要功能相关。

在 Web 服务中使用 SOAP 报头的方法非常简单，主要分为以下几步。

- 创建一个从 SoapHeader 类派生的自定义类，用于封装传入 SOAP 标头的数据。
- 将一个自定义的 SoapHeader 类对象作为成员添加到 Web 服务的类中。
- 在 Web 服务类中的 Web 服务方法中使用 SoapHeader 属性指定自定义的 SOAP 标头的成员类。
- 在 Web 服务方法或 Web 服务客户端中访问 Web 服务方法的时候将 SOAP 报头对象作为请求的第一个参数传入。

要在 Web 服务中使用 SOAP 报头，需要实现自己的 SOAP 报头类。自定义的 SOAP 报头类需要继承 SoapHeader 类(该类在命名空间 System.Web.Services.Protocols 中)。示例代码如下：

```
public class MyHeader : SoapHeader
{
    private string username;
    private string password;
    public string Username
    {
        get{ return username; }
        set{ username = value; }
    }
    public string Password
    {
        get{ return password; }
        set{ password = value; }
    }
}
```

当有了一个 SOAP 报头类以后，可以将其作为一个公有成员声明在 Web 服务类中，如下所示：

```
public class WebService1 : System.Web.Services.WebService
{
    public MyHeader Header;
    //其他代码省略
}
```

这里可以在 Web 服务类中的 Web 服务方法中使用 SoapHeader 来指定消息头所封装的成员，代码如下：

```
[WebMethod]
[SoapHeader("Header")]
public string HelloWorld()
{
    return "Hello World";
}
```

在客户端声明一个创建的 SOAP 报头类的实例，然后在调用这个 Web 服务方法的时候将 SOAP 报头对象作为 Web 服务方法的第一个参数传入，就可以在 Web 服务中直接使用 Web 服务的成员对象访问设置的 SOAP 报头信息了。

10.5.2　实例描述

在 Web 服务的几乎所有验证方法中，使用自定义 SOAP 报头的方式应该是最方便，也最容易理解和实现的一种安全验证方法。

本实例，我们就使用自定义 SOAP 报头的方式对用户信息查询功能进行安全验证。

这个例子中，在 SOAP 报头中保存一个内部人员才知道的口令，如果用户请求对上口令，则可以请求到相应的结果，否则抛出一个异常信息。

10.5.3　实例应用

【例 10-3】 实现自定义验证的 Web 服务

(1) 首先在 Web 服务项目中创建一个 Web 服务，命名为 WSUsers。

(2) 这里需要一个 SOAP 报头类，用于封装报头信息。在 Web 服务项目中创建一个类，命名为 WSUsersKey，该类继承自 SoapHeader 类。在该类中只需要一个属性 Key，用于封装一个只有内部人员才知道的密钥。WSUsersKey 类的代码如下所示：

```
public class WSUsersKey : SoapHeader
{
    private string key;
    public string Key
    {
        get { return key; }
        set { key = value; }
    }
}
```

(3) Web 服务 WSUsers 用于查询用户信息。用户信息中包括用户名、电话号码和邮箱地址等信息，所以我们要设计一个封装用户信息的实体类 UserInfo，代码如下：

```
public class UserInfo
{
    private string username;
    private string phone;
    private string email;
    public string Username
    {
        get { return username; }
        set { username = value; }
    }
    public string Phone
    {
        get { return phone; }
        set { phone = value; }
    }
    public string Email
```

```
    {
        get { return email; }
        set { email = value; }
    }
}
```

(4)　接下来，可以在 Web 服务 WSUsers 类中实现业务逻辑。

这里要实现一个根据用户名查询用户信息的 Web 服务方法 FileUserByName，在执行该方法的时候验证用户传入的 SOAP 消息头中的内容是否是设定的密钥“itzcn.com”。如果是，则执行查询并返回结果；如果不是，则抛出异常信息。

WSUsers 类的实现代码如下所示：

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
public class WSUsers : System.Web.Services.WebService
{
    private WSUsersKey sKey;
    public WSUsersKey SKey
    {
        get { return sKey; }
        set { sKey = value; }
    }
    [WebMethod]
    [SoapHeader("SKey")]
    public UserInfo FineUserByName(string username)
    {
        if (SKey.Key != "itzcn.com")
        {
            throw new SoapException();
        }
        UserInfo user = this.GetUser(username);
        return user;
    }
    private UserInfo GetUser(string username)
    {
        //模拟一批数据供客户端查询
        List<UserInfo> users = new List<UserInfo>();
        users.Add(new UserInfo() { Username = "祝红涛", Email = "zhht@itzcn.com",
         Phone = "0371-6548215" });
        users.Add(new UserInfo() { Username = "赵星", Email = "xing@itzcn.com",
         Phone = "010-12365485224" });
        users.Add(new UserInfo() { Username = "马林林", Email = "linlin@itzcn.com",
         Phone = "130254698775" });
        users.Add(new UserInfo() { Username = "段少治", Email = "
         duanshaozhi@itzcn.com", Phone = "1383812582" });
        foreach (var u in users)
        {
```

```
            if (u.Username == username)
            {
                return u;
            }
        }
        return null;
    }
}
```

完成服务器端后，下面来设计客户端的 Windows 程序。

(5) 在客户端项目中添加对该 Web 服务的服务引用。

(6) 在客户端项目中创建一个 Windows 窗体。该窗体实现输入用户名，查询出该用户的联系方式信息。窗体界面的设计如图 10-14 所示。

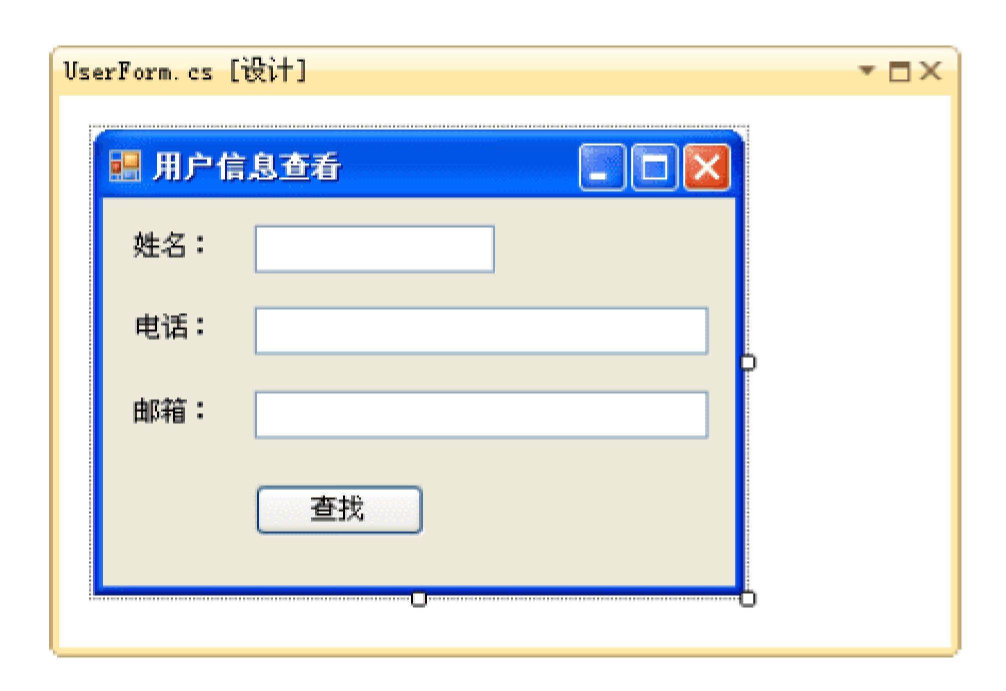


图 10-14　查询用户信息窗体

(7) 现在要做的工作就是在单击【查找】按钮时，根据输入的“姓名”查询用户信息，并将联系方式显示到窗体中。所以，【查找】按钮的单击事件处理程序代码如下所示：

```
private void btnSearch_Click(object sender, EventArgs e)
{
    //创建代理类对象
    ServiceUsers.WSUsersSoapClient wsc = new ServiceUsers.WSUsersSoapClient();
    //创建 SOAP 报头对象
    ServiceUsers.WSUsersKey key = new ServiceUsers.WSUsersKey();
    //设置报头对象内容
    key.Key = "itzcn.com";
    //调用 Web 服务方法，将 SOAP 报头对象作为第一个参数传入
    var user = wsc.FineUserByName(key, this.txtUsername.Text);
    //显示查询结果
    this.txtEmail.Text = user.Email;
    this.txtPhone.Text = user.Phone;
}
```

10.5.4　运行结果

首先使用浏览器访问该 Web 服务，执行以后，服务器端抛出异常，结果如图 10-15 所示。

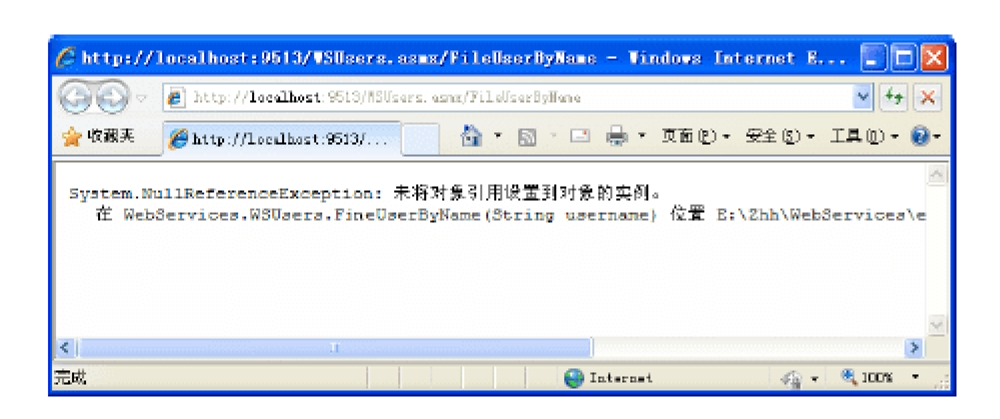


图 10-15　请求异常

这里因为在使用 Web 服务方法进行权限判断时，并没有为其传递 SOAP 消息头对象，而强行使用抛出的空指针异常。



执行以后浏览器默认显示的是 HTTP 500 错误，设置 Internet 高级选项中取消显示友好的错误信息，就会显示出请求详细的错误信息。

接下来使用 Windows 窗体应用程序来测试这个 Web 服务。在“姓名”文本框中输入“段少治”，然后单击“查找”按钮，稍等，系统就会返回相应的用户信息，如图 10-16 所示。



图 10-16　用户信息查看

10.5.5　实例分析



源码解析：

本实例，首先创建一个查询用户信息功能的 Web 服务 WSUsers；然后创建一个用于封装 SOAP 报头信息的类 WSUsersKey，该类继承自 SoapHeader 类；接着在 Web 服务中实现相应的查询功能，并使用 SoapHeader 方法设置 Web 服务方法的报头对象；最后在客户端声明一个刚才创建的 SOAP 报头对象，并作为 Web 服务方法的第一个参数传递给 Web 服务，就可以在服务器端直接访问该 SOAP 报头信息了。

10.6　常见问题解答

10.6.1　如何设计安全的 Web 服务



如何设计安全的 Web 服务？

网络课堂：http://bbs.itzcn.com/thread-15667-1-1.html

刚学了 Web 服务，发现使用起来非常方便，但是它也像 Web 一样是运行在 HTTP 协议之上的。

我们在使用 Web 服务的时候如何才能设计出安全的 Web 服务呢？我应该考虑哪些方面呢？

【解决办法】

设计一个安全的 Web 服务需要具有如下思想：使用尝试或检验过的设计原则来设计 Web 服务项目；集中处理关键区域。

另外还要注意其他一些方面，包括输入验证、身份验证、授权、配置管理、敏感数据保护、会话管理、密码系统、参数处理、异常管理和审核与日志记录等项。另外还要特别注意部署问题，包括拓扑、网络基础设施、安全策略和步骤等。

可能这些项非常杂乱，让你听起来非常头疼。不过你可以从前到后慢慢地了解，一般来说也很少有人能一下子学会所有的东西。

10.6.2 Web 服务的安全体系



谁来给我简单地介绍一下 Web 服务的安全体系？

网络课堂：http://bbs.itzcn.com/thread-15668-1-1.html

刚学 Web 服务，想深入了解一下关于 Web 服务的安全性问题。谁来给我简单地介绍一下 Web 服务的安全体系？

【解决办法】

整个 Web 服务的安全性设计需要考虑以下三个级别。

(1) 平台/传输级安全性。

平台/传输级安全性是指 Internet 中的两个平台之间的安全传输通道，它连接 Web 服务的客户端和服务器端，在平台与平台之间进行一个点对点的安全过滤。它的特点是简单明了。除了能在 Web 服务中使用外，还可以应用在其他各种 Internet 互联网应用程序解决方案中。它的缺点是和应用平台集成过于紧密，性能取决于基本的平台、网络传输机制和安全性服务提供程序。

(2) 应用程序级安全性。

应用程序级安全性是指由应用程序负责提供的安全性机制，所有的安全性控制全部由应用程序自己定义。

(3) 消息级安全性。

消息级安全性是指由网络中发送的消息本身来控制数据的安全性，这种方式非常灵活。

另外还要注意的一点是，这三种级别的安全性是针对 Web 服务以及其所处环境相关的几个方向的设计，有时候需要让它们三个协同工作才能构建完美的 Web 服务应用程序。

10.7　习　　题

一、填空题

(1) 与 Web 服务有关的安全选项中支持多种类型的安全选项，包括 SSL、简要验证、__________、客户授权证书验证、自定义验证。

(2) 在 Internet 中，对于整个 Web 服务的安全性设计需要考虑三个级别：平台/传输级(点对点)安全性、应用程序级(自定义)安全性和__________。

(3) IP 安全是一种 Internet 标准，用于保护在__________层传输的 Internet 数据流量。

(4) 在 ASP.NET Web 服务应用程序中，基本验证、简要验证和集成 Windows 验证方法作为一项服务由__________提供给 Web 服务。

二、选择题

(1) 下面说法错误的是__________。

A. 验证是一种对客户端用户的权限进行检验的机制。它可以接收用户的身份证，并进行用户权限的检验。

B. 权限通常是一个确认的信息，标示着该用户在应用程序中的可操作范围。

C. 授权是客户端授予本机用户访问某个服务器站点中的资源的一种处理过程。

D. 加密是一种信息发送端和信息接收端协商制定的一种规则。我们可以使用它来进行数据转换，以确保数据的实际信息只能由信息发送端和信息接收端知道。

(2) 下列选项中，不是使用表单验证的优点的一项是__________。

A. 易于实现

B. 无特殊要求

C. 支持自定义权限

D. 可以使用操作系统中的账户进行验证

E. 灵活性强

(3) 默认情况下，Web 服务客户端可以使用三种方式与 ASP.NET Web 服务进行通信，下面不属于这三种方式中的一项是__________。

A. HTTP GET

B. HTTP POST

C. HTTP SOAP

D. HTTP PUT

(4) 下面选项不属于使用 Windows 验证的优点的一项是__________。

A. 容易实现。Windows 验证基于 Windows 系统的安全机制，使用起来不需要太多额外的技术。

B. 能完美地被 Web 服务器支持。Windows 验证可以与任何系统平台中的账户进行集成使用。

C. 无过高要求。配置起来非常简单，只需要在 Windows 视窗界面中单击几下鼠标即可完成配置。

D. 密码从不通过网络发送。Windows 验证会在客户端将用户密码使用散列算法进行加密，因此保证了密码的安全性。

三、上机练习

上机练习 1： 在 Web 服务项目中配置使用 Windows 验证。

本次练习要求以例 10-3 中的查询用户信息的 Web 服务 WSUsers 为例，将其发布成一个 Web 项目，并上传到 Web 服务器中。然后在 Web 服务器中将其访问权限验证方式修改为“集成 Windows 身份验证”。

设置以后使用浏览器请求该 Web 服务，运行结果如图 10-17 和图 10-18 所示。

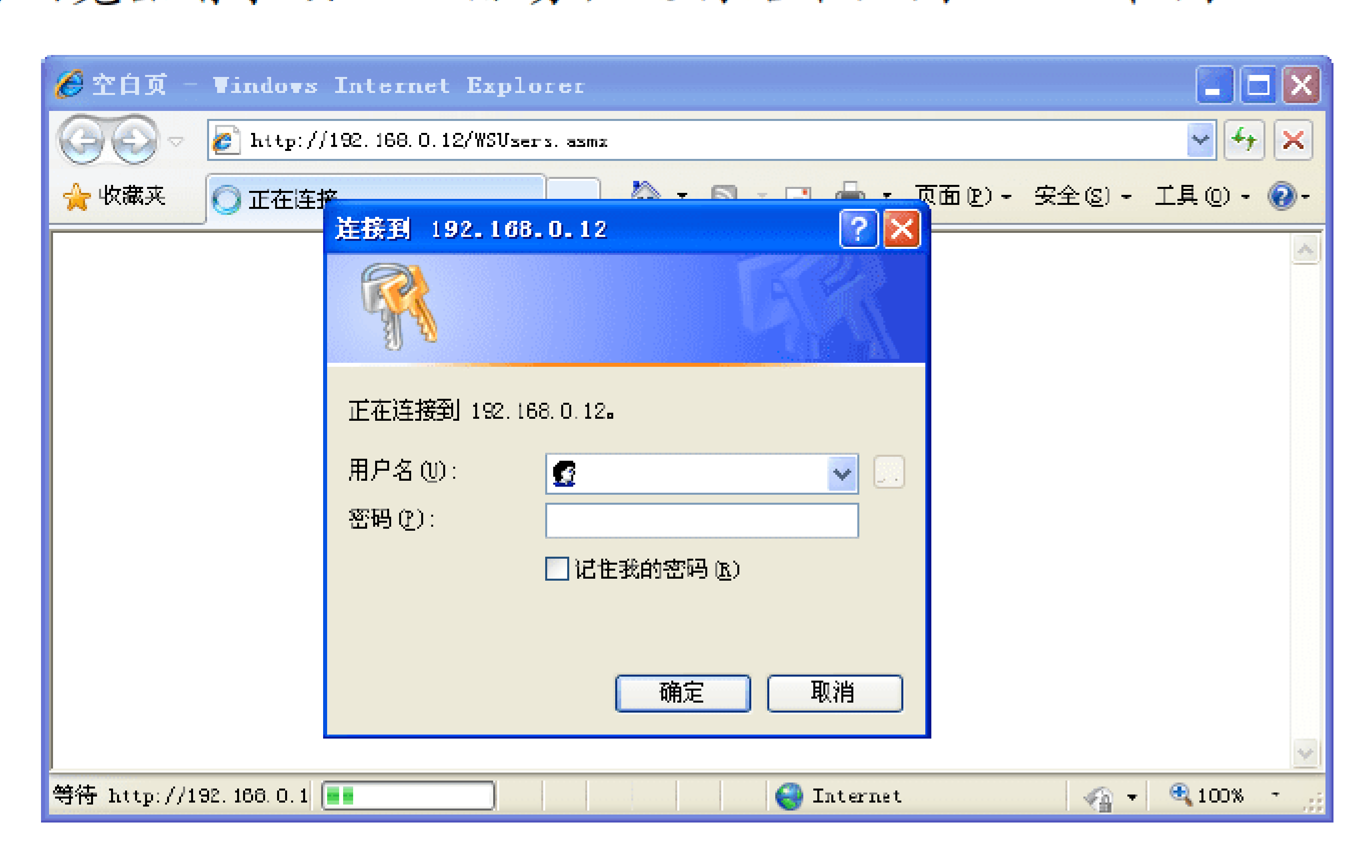


图 10-17　登录提示框

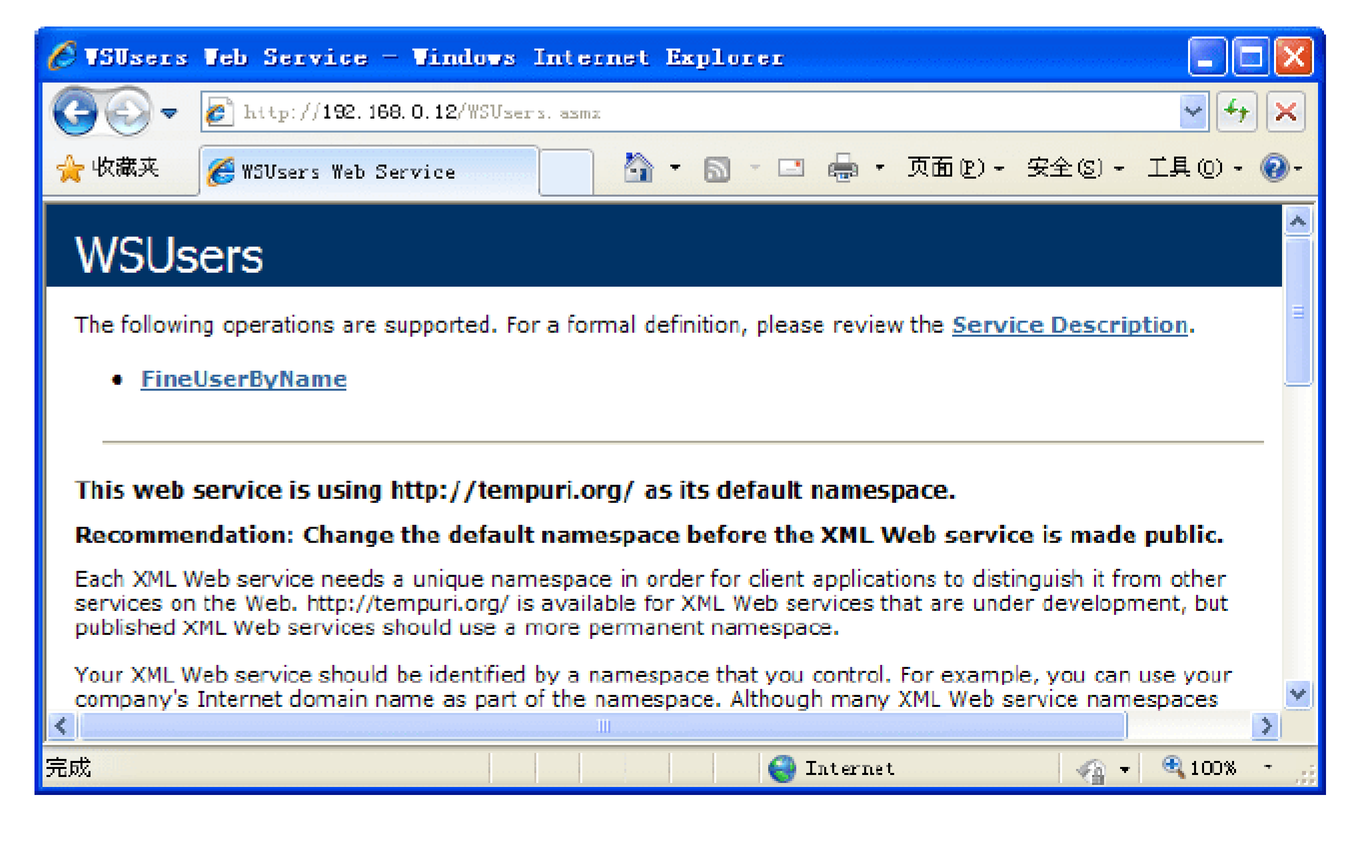


图 10-18　验证通过



第 11 章　.NET 下的 XML 操作

内容摘要：

XML(Extensible Markup Language，可扩展性标记语言)是一种描述数据和数据结构的语言，是基于 Web 数据交互的语言。XML 极其简单且易于掌握和使用，又因为其跨平台的特性，而被广泛应用。

本章将学习如何在 ASP.NET 中读取和写入 XML，以及如何对 XML 中的属性和节点进行增删改的操作。同时，还学习了如何对 XML 进行序列化。

学习目标：

- 掌握 4 种读取 XML 的方式
- 掌握两种写入 XML 的方式
- 会使用 DOM 技术对 XML 文件进行增删改操作
- 了解自定义 XML 序列化

11.1　从 XML 文件中读取新闻

在 ASP.NET 中读取 XML 文档信息主要有如下 4 种方式，本节将对它们进行详细介绍。

- 使用 XML 控件读取。
- 使用 DOM 技术读取。
- 使用 DataSet 对象读取。
- 使用 XmlTextReader 类读取。

**视频教学：**光盘/videos/11/从 XML 文件中读取新闻.avi　　长度：13 分钟

11.1.1　基础知识——读取 XML

现在，我们新建一个 ASP.NET 项目，在项目中添加一个 XML 文件。然后，用 4 种方式读取 XML 文档的信息，XML 文档的代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<teachers>
    <teacher>
        <name>张华</name>
        <age>32</age>
        <teach>化学</teach>
        <address>河南省郑州市</address>
    </teacher>
    <teacher>
        <name>李江</name>
        <age>45</age>
        <teach>物理</teach>
        <address>河南省信阳市</address>
    </teacher>
    <teacher>
        <name>王丽丽</name>
        <age>语文</age>
        <teach>26</teach>
        <address>山东省济南市</address>
    </teacher>
    <teacher>
        <name>齐菲菲</name>
        <age>28</age>
        <teach>数学</teach>
        <address>湖南省长沙市</address>
    </teacher>
</teachers>
```

1. 使用 XML 控件读取

使用 XML 控件读取是比较简单的一种方式，XML 控件有一个名为 DocumentSource 的属性，我们只需要将所要读取的 XML 文档的路径赋给该属性即可。

在项目中添加一个名为 UseXML.aspx 的页面，向页面中添加如下代码：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>使用 XML 控件读取</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Xml ID="Xml1" runat="server"
DocumentSource="~/Teacher.xml"></asp:Xml>
    </div>
    </form>
</body>
</html>
```

可以直接从【工具箱】中将 XML 控件拖至 UseXML.aspx 页面。运行该页面，在浏览器中查看运行结果，如图 11-1 所示。

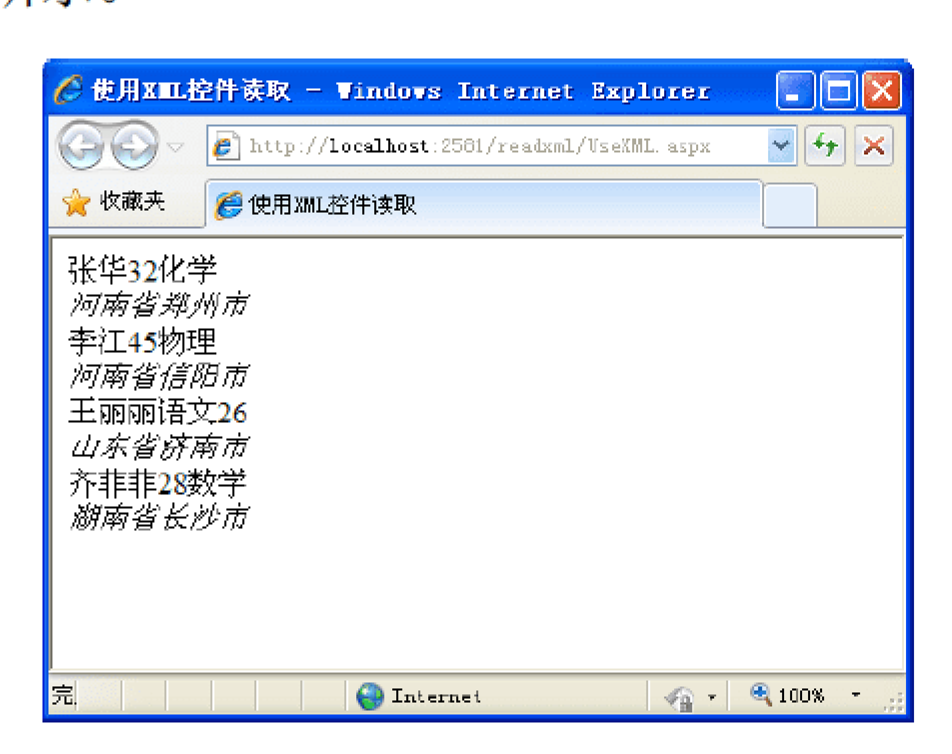


图 11-1　使用 XML 控件读取

该方法只需要将 XML 文件的路径赋给 XML 控件的 DocumentSource 属性就可以了，方法很简单，可是数据输出的格式却不尽如人意。

2. 使用 DOM 技术读取

.NET Framework 的 XML 类为我们提供了一个符合 W3C DOM 标准的 XML 分析器对象 XmlDocument，它是在.NET 环境中执行大多数基于 XML 操作的核心对象。通过下面的例子了解怎样使用 XmlDocument 对象读取 XML 文档。

新建一个名为 UseDOM.aspx 的页面，然后添加一个 XML 控件，我们不需要设置控件的任何属性。代码如下所示：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="UseDOM.aspx.cs"
Inherits="UseDOM" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>使用 DOM 技术读取</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Xml ID="Xml1" runat="server"></asp:Xml>
    </div>
    </form>
</body>
</html>
```

然后打开 UseDOM.aspx 页面的 cs 文件 UseDOM.aspx.cs，向文件中添加如下代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;
public partial class UseDOM : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        XmlDocument dom = new XmlDocument();
        dom.Load(Server.MapPath("teache.xml"));
        Xml1.Document = dom;
    }
}
```

运行 UseDOM.aspx 页面，结果如图 11-2 所示。

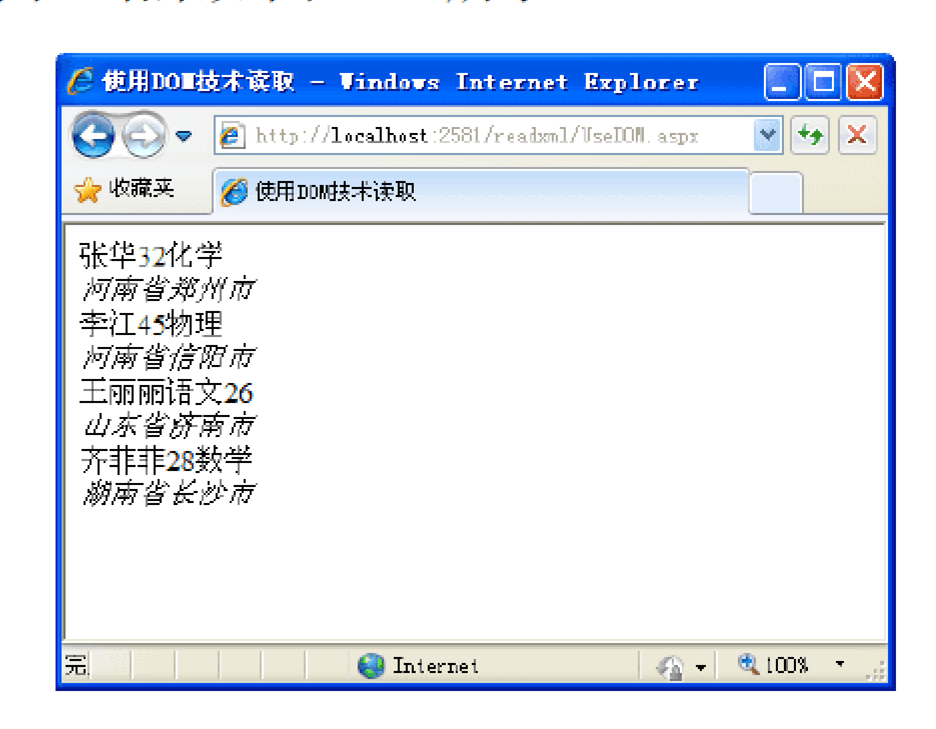


图 11-2　使用 DOM 技术读取

从图 11-2 中可以看到，使用 DOM 技术读取与使用 XML 控件读取信息显示的结果相同。虽然我们同样使用了 XML 控件，但是 DOM 技术并没有使用 DocumentSource 属性，而是使用

了 XmlDocument 对象的 Load 方法载入要读取的 XML 文档，然后将 XML 控件和 XmlDocument 对象关联起来。

3．使用 DataSet 对象读取

DataSet 将数据和架构作为 XML 文档格式进行读写。如表 11-1 所示，列出了 DataSet 处理 XML 的常用方法。

表 11-1　DataSet 处理 XML 的常用方法

方　法	说　明
GetXml	返回存储在 DataSet 中的数据的 XML 表示形式
GetXmlSchema	以 XML 形式返回存储在 DataSet 中的 XSD 架构
ReadXml	用于将 XML 架构和数据读入 DataSet
ReadXmlSchema	用于将 XML 架构读入 DataSet
WriteXml	用于将 DataSet 的数据写入 XML 中
WriteXmlSchema	用于写 XML 架构形式的 DataSet 结构

下面通过实例详细描述如何使用 DataSet 对象载入 XML 文档。

首先，在项目中添加一个名为 UseDataSet.aspx 的页面，并在文件中添加如下代码：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="UseDataSet.aspx.cs"
Inherits="UseDataSet" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:GridView ID="GridView1" runat="server">
      </asp:GridView>
   </div>
   </form>
</body>
</html>
```

然后，打开 UseDataSet.aspx 页面的 cs 文件 UseDataSet.aspx.cs，并添加如下代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
public partial class UseDataSet : System.Web.UI.Page
```

```
{
    protected void Page_Load(object sender, EventArgs e)
    {
        DataSet ds = new DataSet();
        ds.ReadXml(Server.MapPath("Teacher.xml"));
        GridView1.DataSource = ds.Tables["teacher"].DefaultView;
        GridView1.DataBind();
    }
}
```

运行 UseDataSet.aspx 页面，运行结果如图 11-3 所示。

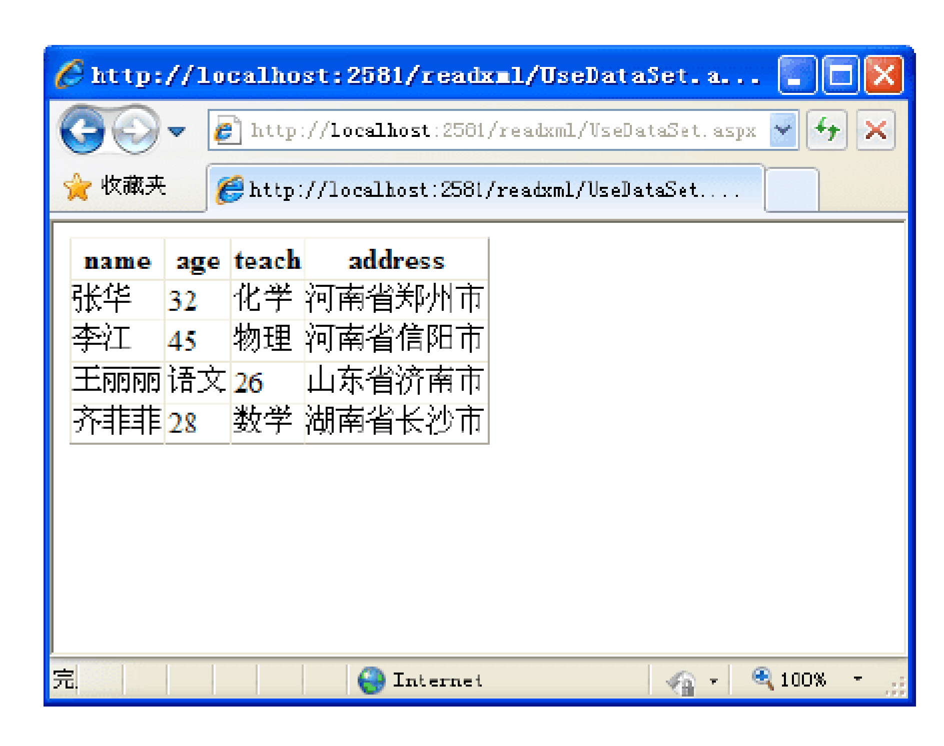


图 11-3　使用 DataSet 对象载入 XML 文档

该例中，先使用 DataSet 对象载入 XML 中的数据，然后使用 GridView 控件将数据显示在页面上。数据以表格形式展示，显然比之前两种更清晰。

4．使用 XmlTextReader 类读取

使用 XmlTextReader 对象可以读取磁盘文件，并且以 XML 节点列表形式显示数据。下面举例说明如何使用 XmlTextReader 对象读取 XML 文件。因为是以文本形式读取，所以前台页面 UseText.aspx 很简单，我们只需要放一个 Label 控件就可以了，Label 控件名为 lblXML。下面是 UseText.aspx 页面的后台代码：

```
protected void Page_Load(object sender, EventArgs e)
    {
        XmlTextReader xtxtR = new
XmlTextReader(Server.MapPath("Teacher.xml"));
        string strResult = "";
        XmlNodeType xnt;
        while (xtxtR.Read())
        {
            xnt = xtxtR.NodeType;
            switch (xnt)
            {
                case XmlNodeType.XmlDeclaration:
                    //读取文件头
                    strResult += "<font color='#0033FF'>文件头:</font>" +
                        xtxtR.Name + "" + xtxtR.Value + "</br>";
```

```
                break;
            case XmlNodeType.Element:
                //读取标签
                strResult += "<font color='#0099FF'>读取标签:</font>" +
                    xtxtR.Name + "" + xtxtR.Value + "</br>";
                break;
            case XmlNodeType.Text:
                //读取值
                strResult += " -<font color='#00CCFF'>值:</font>" +
                    xtxtR.Name + "" + xtxtR.Value + "</br>";
                break;
        }
        if (xtxtR.AttributeCount > 0)
        {
            while (xtxtR.MoveToNextAttribute())
            {
                strResult += " <font color='#ff3dee'>-属性</font>" +
                    xtxtR.Name + "<font color='#ff3dee'>-值</font>"+xtxtR.
                    Value+"</br>";
            }
        }
        lblXML.Text = strResult;
    }
}
```

在本例中，我们使用 XmlTextReader 对象创建实例，读取 XML 文件，显示的结果以节点列表形式输出，如图 11-4 所示。

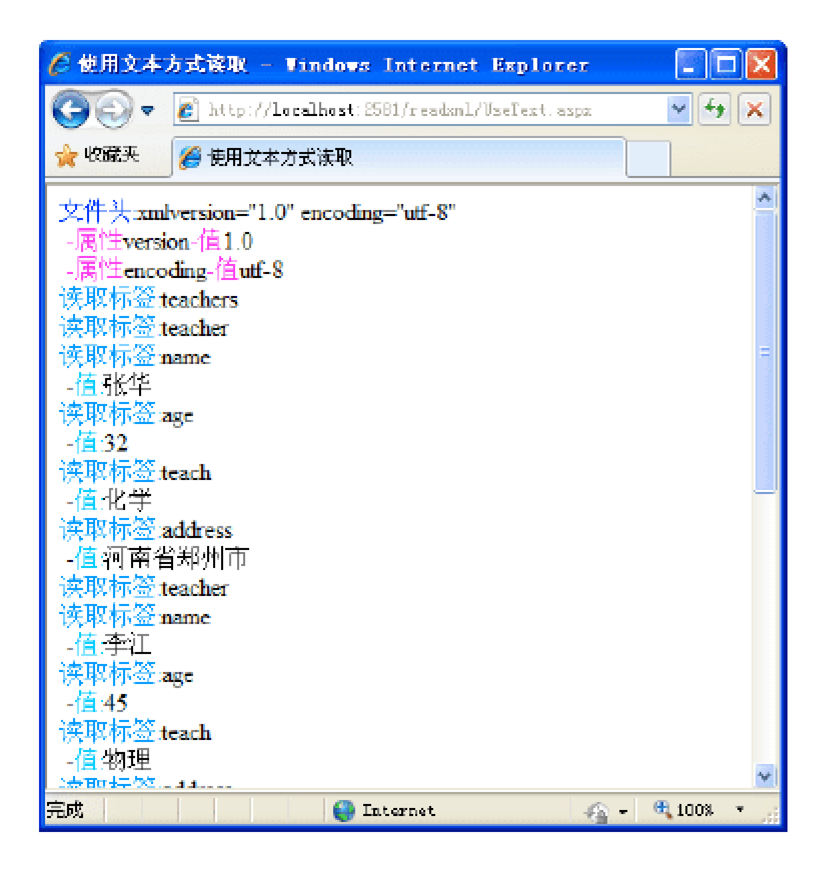


图 11-4　使用 XmlTextReader 读取 XML 文件

11.1.2　实例描述

我们已经了解到读取 XML 的 4 种方式，根据 4 种方法显示的结果来看，使用 DataSet 方式读取显示的结果是以表格形式显示，条理分明、结构清晰。

下面通过一个保存新闻信息的 XML 文档，看看如何使用 DataList 控件进行读取并显示。

11.1.3 实例应用

【例 11-1】 从 XML 文件中读取新闻

(1) 打开 11.1.2 节建立的 ASP.NET 项目，然后新建一个 XML 文件，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<news>
    <new>
        <ID>001</ID>
        <title>花样衬衫巧搭修身裤 魅力无法挡</title>
        <image>1.jpg</image>
        <content>衬衫一向以一种独特的中性美取胜，而本季的衬衫却大相径庭——它更为妩媚与柔
        美，独特的设计为都市女性树立了独立间不失性感与优雅的气质。巧搭配裤装，享受这个多姿
        多彩的早春气息吧。时尚解析： 阔腿裤永远走在时尚前沿，拥有了大量追捧者。</content>
    </new>
    <new>
        <ID>002</ID>
        <title>健康指南：红薯千万别跟什么同吃</title>
        <image>2.jpg</image>
        <content>喜欢吃杂粮的刘大爷告诉记者，前几天一次晚饭时吃了几个红薯，相隔不到半小时
        又吃了一个柿子，晚上看电视的时候肚子开始疼了。 红薯忌与柿子同吃 红薯和柿子不宜在短
        时间内同时食用，如果食量多的情况下，应该至少相隔五个小时以上。</content>
    </new>
    <new>
        <ID>003</ID>
        <title>老人早春饮食，一二三四口诀</title>
        <image>3.jpg</image>
        <content>不宜硬：在早晨，老年人不宜进食油腻、煎炸、干硬以及刺激性大的食物，否则会
        劳脾伤胃，导致食滞于中，消化不良。老年人早餐宜吃容易消化的温热、柔软食物，如加些莲
        子、红枣、山药、桂圆和薏仁等保健食品，则效果更佳。</content>
    </new>
    <new>
        <ID>004</ID>
        <title>iPhone 再现全民漂移 跑跑卡丁车口袋版</title>
        <image>4.jpg</image>
        <content>泡泡网手机频道 3 月 15 日 红遍全国的网络游戏“跑跑卡丁车”如今被完美移植
        到了 iPhone，短短 1 天时间便以免费的实惠与精美的制作跻身排行榜榜首。</content>
    </new>
</news>
```

(2) 新建一个名为 ShowNews.aspx 的页面，在页面内添加一个 DataList 控件，页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ShowNews.aspx.cs"
Inherits="ShowNews" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>实例-显示新闻列表</title>
</head>
<body>
   <form id="form1" runat="server">
   <div style="font-family: 宋体_GB2312; font-size: 12px;">
      <asp:DataList ID="DataList1" runat="server">
         <ItemTemplate>
            <table border="0" width="760" id="table1" cellspacing="0"
             cellpadding="0" bgcolor="#ffffff">
               <tr>
                  <td width="150" align="center">
                     <img border="0" src='image/<%#Eval("image") %>'
                      width="110" height="120" align="center">
                  </td>
                  <td style="line-height: 1" valign="top">
                     <br>
                     <%#Eval("ID") %><br>
                     标题: <%#Eval("title") %><br>
                     内容: <%#Eval("content") %><br>
                  </td>
               </tr>
            </table>
         </ItemTemplate>
      </asp:DataList>
   </div>
   </form>
</body>
</html>
```

(3) ShowNews.aspx 页面的后台代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
   DataSet ds = new DataSet();
   ds.ReadXml(Server.MapPath("News.xml"));
   DataList1.DataSource = ds.Tables[0].DefaultView;
   DataList1.DataBind();
}
```

(4) 保存，这样实例就完成了。

11.1.4 运行结果

运行 ShowNews.aspx，显示结果如图 11-5 所示。



图 11-5　运行结果

11.1.5　实例分析



源码解析：

在实例中，我们使用 DataList 数据控件从 XML 中读取数据和显示。其实，ASP.NET 中的任意一个数据控件都可以完成，但是要注意需要在前台页面使用 Eval 进行数据绑定，否则数据将无法呈现。

11.2　写入 XML 的收件箱

我们已经掌握了怎样在.NET 中读取 XML，本节将学习如何向 XML 中写入内容。这些内容可以是在创建时指定的，也可以手动一次写入一个指定的 XML 元素。选择哪种方式完全取决于读者的喜好。



视频教学：光盘/videos/11/写入 XML 的收件箱.avi　　长度：15 分钟

11.2.1　基础知识——写入 XML

在 ASP.NET 中写入 XML 文件有很多种方法，在这里只介绍两种比较常用的方法：使用 DataSet 写入和使用文本方式写入。

使用 DataSet 和文本方式写入 XML 不同的是：使用 DataSet 方式写入 XML 文档所需的数据是从数据库表中直接读取的；而使用文本方式写入 XML 则是手动写入。下面我们就来详细

了解两种写入 XML 的方法。

1. 使用 DataSet 写入

有时候我们需要对数据表里的数据进行操作，为了让数据表的操作性更加灵活方便，可以使用 DataSet 对象将数据表里的数据转换为 XML 文件。

要想写入 XML 文件，只需要使用 DataSet 对象的 WriteXml 方法。在 System.Data 命名空间下的 DataSet 对象可以很容易地将一个数据库文件存储为 XML 文件。具体步骤如下。

(1) 先准备好要写入 XML 文件的数据库表，在这里使用 myDataBase 数据库的 Users 表，我们使用的数据库是 SQL Server 2008，Users 表的表结构和数据如图 11-6 所示。

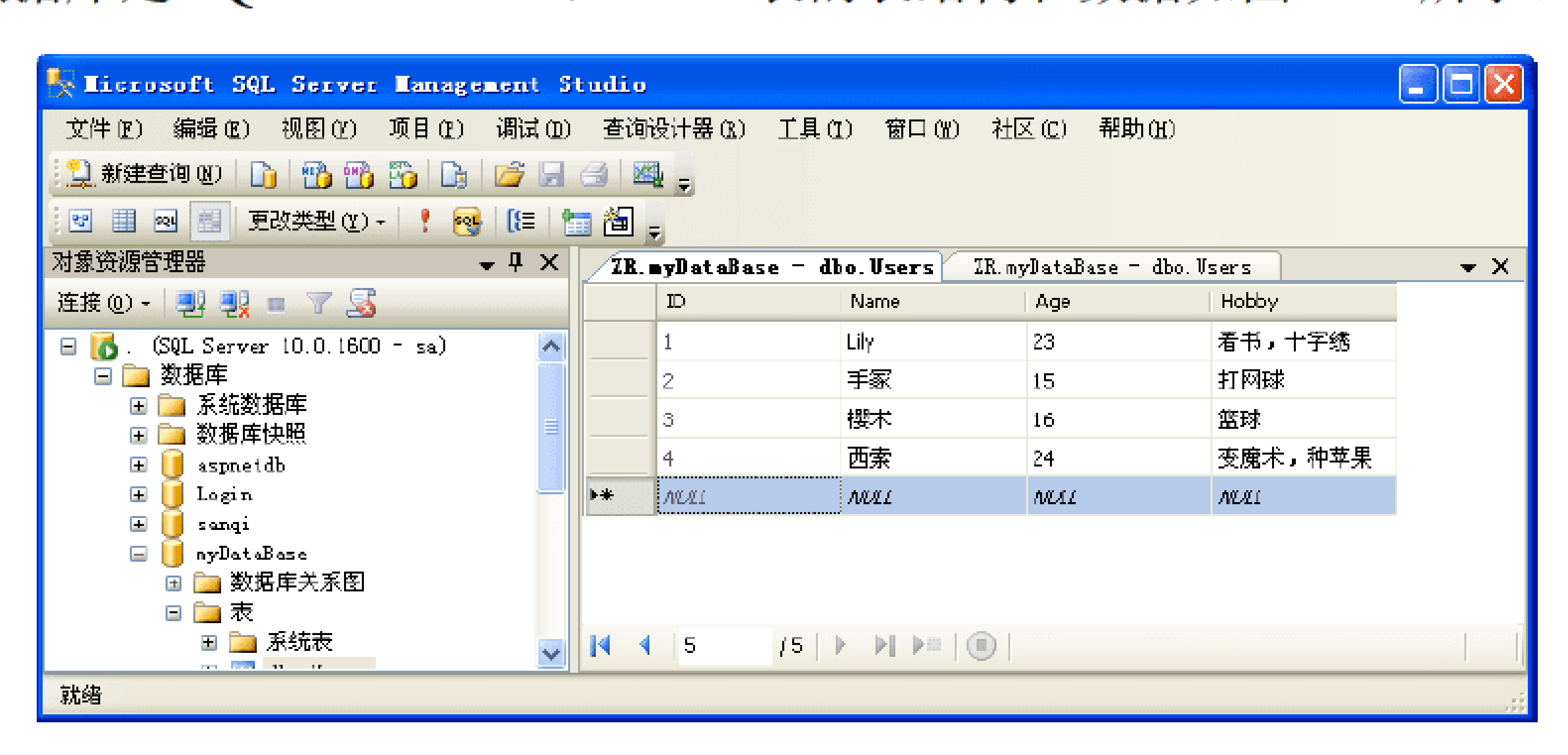


图 11-6 Users 表的表结构和数据

(2) 新建一个 ASP.NET 项目，项目名称为 writerxml，添加 UserDataSet.aspx 页面，然后在 UserDataSet.aspx 页面的后台文件 UserDataSet.aspx.cs 里添加如下代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;
using System.Data;

public partial class _UserDataSet: System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      string conString = "Data Source=.;Initial Catalog=myDataBase;User
       ID=sa;Password=123";
      SqlConnection con = new SqlConnection(conString);
      string strSql = "select * from Users";
      try
      {
         con.Open();
         SqlDataAdapter sda = new SqlDataAdapter(strSql, con);
```

```
            DataSet ds = new DataSet();
            sda.Fill(ds);
            ds.WriteXml(Server.MapPath("Users.xml"));

            Response.Write("Users.xml 文件创建成功！");
        }
        catch (SqlException ex)
        {
            Response.Write(ex.Message);
        }
        finally
        {
            con.Close();
        }
    }
}
```

(3) 运行该页面，浏览器显示结果，如图 11-7 所示。这表示 XML 文件创建成功。

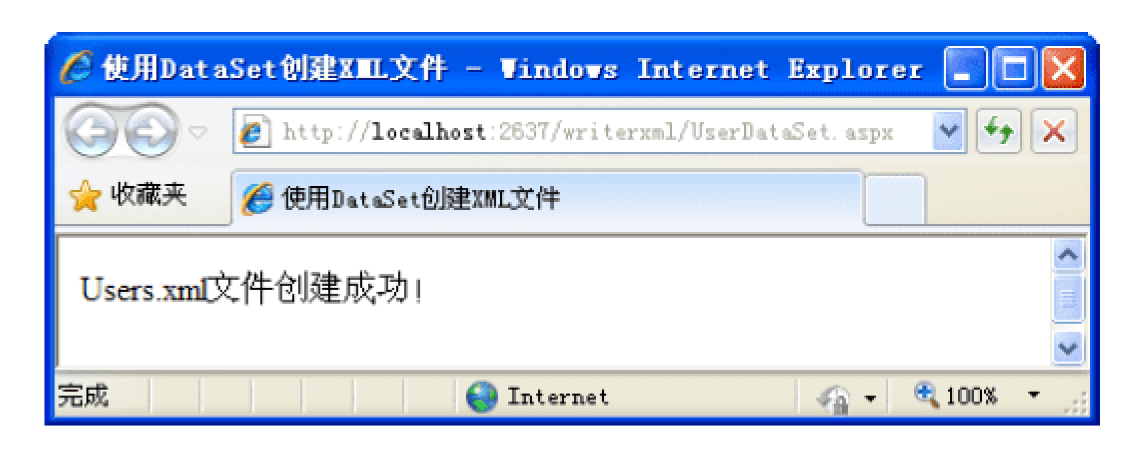


图 11-7 XML 文件创建成功

(4) 在【解决方案资源管理器】中刷新网站目录，创建好的 XML 文件就会出现在当前目录中。在浏览器中打开创建好的 Users.xml 文件，可以看到写入的内容，如图 11-8 所示。

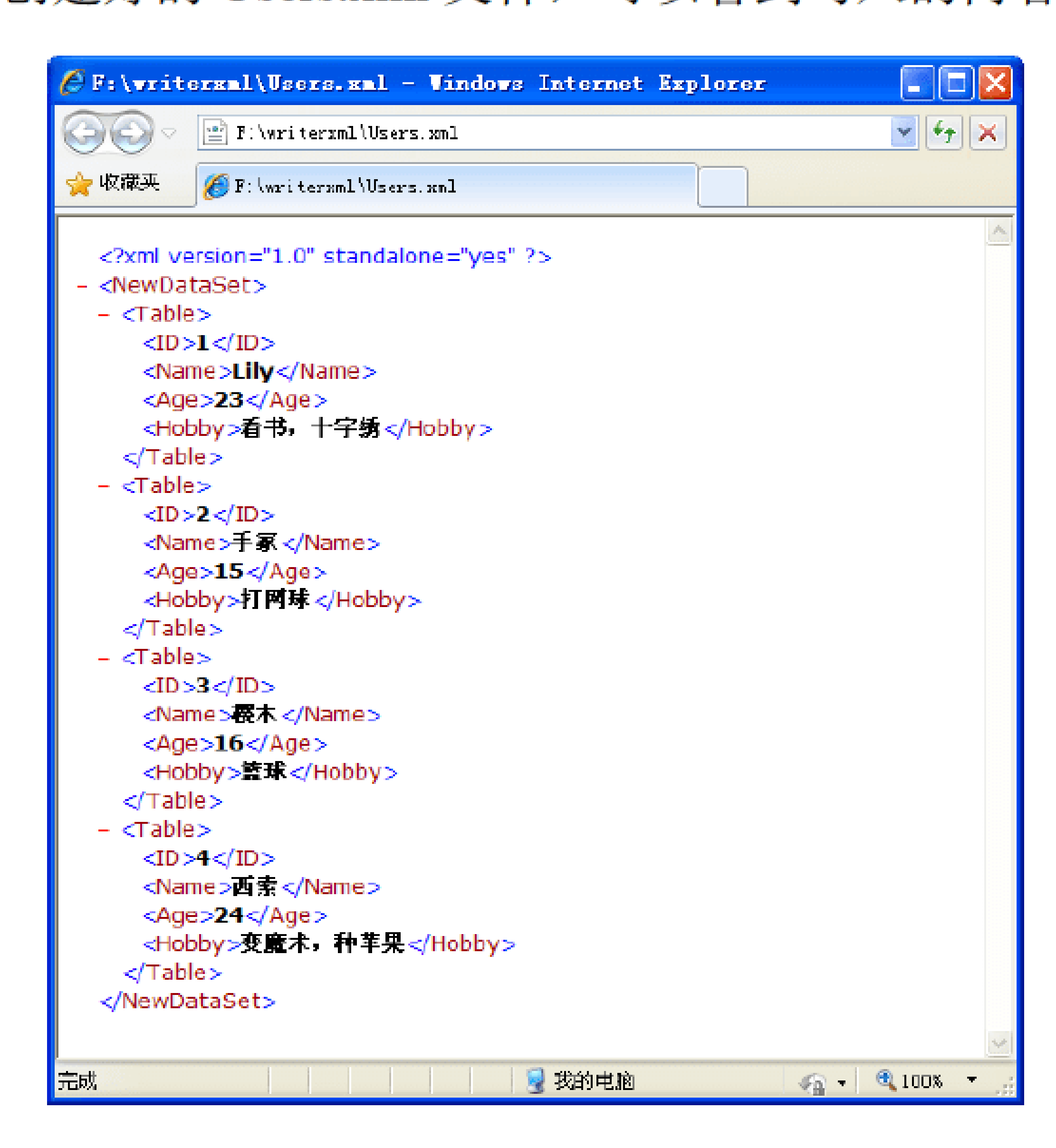


图 11-8 XML 文件内容

2．使用文本方式写入

在 11.1 节我们使用 XmlTextReader 类从磁盘上读取文件。与它对应的 XmlTextWriter 类可以完成 XML 文件的写入操作。下面是使用 XmlTextWriter 类的具体步骤。

(1) 打开 11.1 节创建好的项目，添加一个名为 UserText.aspx 的页面，打开 UserText.aspx 页面的 UserText.aspx.cs 文件，添加如下代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;

public partial class UserText : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      XmlTextWriter xmlTw = null;
      xmlTw = new XmlTextWriter(Server.MapPath("books.xml"), null);
      xmlTw.Formatting = Formatting.Indented;
      xmlTw.Indentation = 3;
      xmlTw.WriteStartDocument();
      xmlTw.WriteComment("已经使用 XMLTextWriter 创建完毕一" + DateTime.Now);
      xmlTw.WriteStartElement("books");
      xmlTw.WriteStartElement("book");
      xmlTw.WriteAttributeString("Category", "古典名著");
      xmlTw.WriteElementString("Title", "红楼梦");

      DateTime edition = new DateTime(2010, 03, 05);//出版版本
      xmlTw.WriteElementString("edition", edition.ToString("yyyy-MM-dd"));
      int intSales = 32;
      xmlTw.WriteElementString("price",intSales.ToString("G"));
      xmlTw.WriteStartElement("AuthorList");
      xmlTw.WriteElementString("Author", "曹雪芹");
      xmlTw.WriteElementString("Author", "高鹗");
      xmlTw.WriteEndElement();
      xmlTw.WriteEndElement();
      xmlTw.Flush();
      xmlTw.Close();
      Response.Write("已经创建 XML 文件 books.xml");
   }
}
```

如上述代码所示，这里主要是使用 XmlTextWriter 类来完成 XML 的写入，其中用到了 XmlTextWriter 类的大量方法。在表 11-2 中给出了这些方法的说明。

表 11-2 XmlTextWriter 类的方法

方　法	说　明
WriteStartDocument	描述版本为“1.0”的 XML 声明
WriteEndDocument	关闭任何打开的元素或属性
Close	关闭流
WriteDocType	写出具有指定名称和可选属性的 DOCTYPE 声明
WriteStartElement	写出指定的开始标记
WriteEndElement	关闭一个元素
WriteFullEndElement	关闭一个元素，并且总是写入完整的结束标记
WriteElementString	写出包含字符串值的元素
WriteStartAttribute	书写属性的起始内容
WriteEndAttribute	关闭上一个 WriteStartAttribute 调用
WriteRaw	手动书写原始标记
WriteString	书写一个字符串
WriteAttributeString	出具有指定值的属性
WriteCData	写出包含指定文本的 <![CDATA[...]]> 块
WriteComment	写出包含指定文本的注释 <!--...-->
WriteWhiteSpace	写出给定的空白
WriteProcessingInstruction	写出在名称和文本之间带有空格的处理指令，如下所示：<?name text?>
Flush	将缓冲区中的所有内容刷新到基础流，并同时刷新基础流

(2) 运行该页面，结果如图 11-9 所示，表示使用文本创建的 XML 文件已经创建成功。

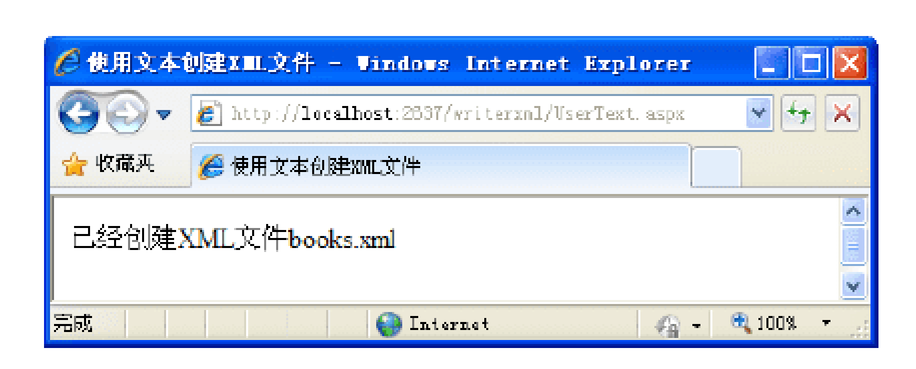


图 11-9 使用文本创建 XML 文件

(3) 在【解决方案资源管理器】中刷新网站目录，创建好的 books.xml 文件已出现在当前目录。在浏览器中打开该文件，显示结果如图 11-10 所示。

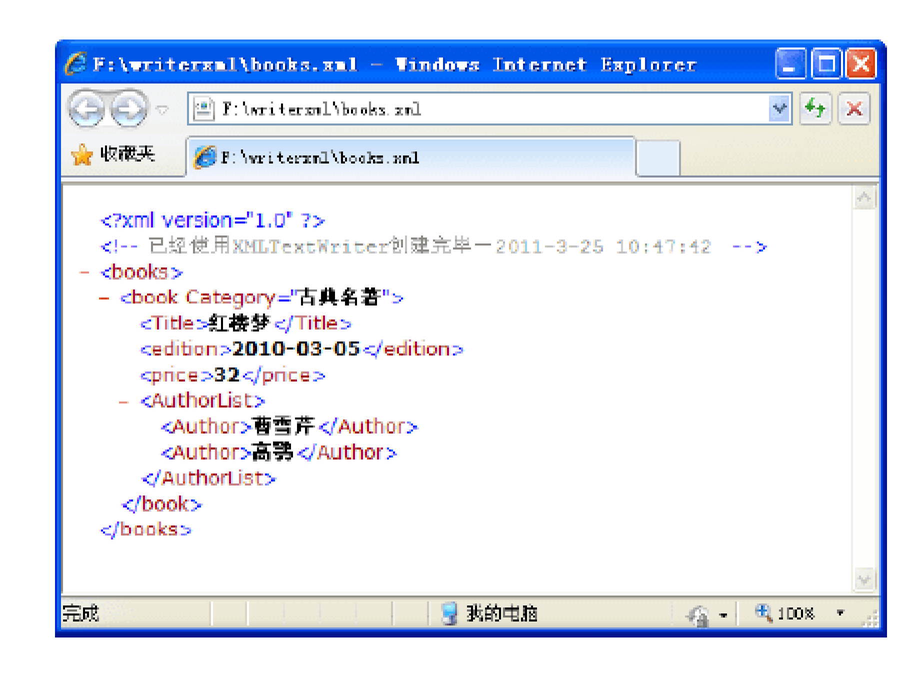


图 11-10 使用文本创建的 XML 文件

11.2.2 实例描述

本节将通过一个实例帮助读者更好地掌握如何写入 XML。此实例使用 XmlTextWriter 类创建一个 XML 文件，再使用 DataSet 进行读取，然后将创建好 XML 文件内容显示在页面上。

11.2.3 实例应用

【例 11-2】 写入 XML 的收件箱

(1) 打开 11.2.1 节创建的项目 writerxml，新建一个名为 message.aspx 的页面，并添加一个 GridView 控件，用于显示数据。前台代码如下：

```
<div>
    <br />
     我的收件箱<br />
     <asp:GridView ID="GridView1" runat="server" BackColor="White"
           BorderColor="#CCCCCC" BorderStyle="None" BorderWidth="1px"
              CellPadding="3"
           EnableModelValidation="True">
           <FooterStyle BackColor="White" ForeColor="#000066" />
          <HeaderStyle BackColor="#006699" Font-Bold="True" ForeColor="White" />
           <PagerStyle BackColor="White" ForeColor="#000066"
              HorizontalAlign="Left" />
           <RowStyle ForeColor="#000066" />
           <SelectedRowStyle BackColor="#669999" Font-Bold="True"
              ForeColor="White" />
     </asp:GridView>
</div>
```

(2) 打开 message.aspx 页面的 cs 文件 message.aspx.cs 文件，添加如下代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;
using System.Data;
public partial class Message : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (GetXML())
        {
            DataSet ds = new DataSet();
            ds.ReadXml(Server.MapPath("message.xml"));
```

```
            GridView1.DataSource = ds.Tables["message"].DefaultView;
            GridView1.DataBind();
        }
    }
    private bool GetXML()
    {
        XmlTextReader xr = new XmlTextReader(Server.MapPath("message.xml"));
        if (xr != null)
        {
            XmlTextWriter xmlTw = null;
            xmlTw = new XmlTextWriter(Server.MapPath("message.xml"), null);
            xmlTw.Formatting = Formatting.Indented;
            xmlTw.Indentation = 3;
            xmlTw.WriteStartDocument();
            xmlTw.WriteComment("已经使用XMLTextWriter创建完毕一" + DateTime.Now);
            xmlTw.WriteStartElement("messages");
            xmlTw.WriteStartElement("message");
            xmlTw.WriteAttributeString("Type", "陌生消息");
            xmlTw.WriteElementString("ID", "0010025");
            xmlTw.WriteElementString("Title", "加我");
            xmlTw.WriteElementString("Content", "我是嘻嘻鼠，加我好友吧");
            DateTime edition = new DateTime(2010, 05, 05);
            xmlTw.WriteElementString("datetime", edition.ToString("yyyy-MM-dd"));
            xmlTw.WriteElementString("FromName", "嘻嘻鼠");
            xmlTw.WriteElementString("ToName", "高山");
            xmlTw.WriteEndElement();
            xmlTw.WriteStartElement("message");
            xmlTw.WriteAttributeString("Type", "系统消息");
            xmlTw.WriteElementString("ID", "0010026");
            xmlTw.WriteElementString("Title", "系统消息");
            xmlTw.WriteElementString("Content", "您的收件箱已满，请查阅并整理");
            DateTime edition1 = new DateTime(2010, 12, 13);
            xmlTw.WriteElementString("datetime", edition1.ToString("yyyy-MM-dd"));
            xmlTw.WriteElementString("FromName", "服务器");
            xmlTw.WriteElementString("ToName", "高山");
            xmlTw.WriteEndElement();
            xmlTw.WriteEndElement();
            xmlTw.Flush();
            xmlTw.Close();
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

GetXML 方法用于创建 XML 文件 message.xml，在创建之前，先使用 XmlTextReader 判断是否存在该 XML 文件，如果不存在，再进行创建。在页面的 Load 事件里，调用方法 GetXML，并且使用 GridView 控件进行显示。



我们在创建 XML 文件时，如果想要添加多条数据，需要在完成一条数据创建后使用 WriteEndElement()关闭这个元素。

11.2.4　运行结果

运行 message.aspx 页面，显示结果如图 11-11 所示。

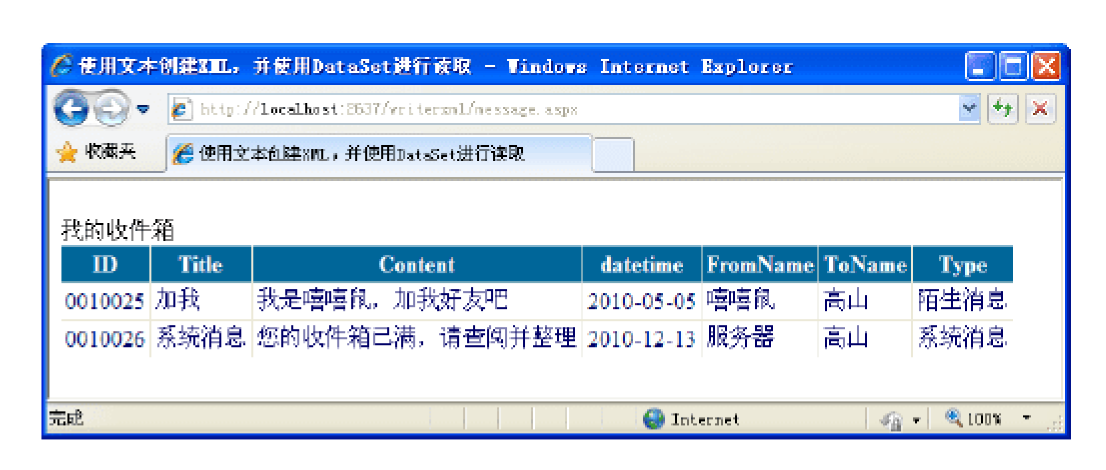


图 11-11　页面显示的结果

打开创建好的 XML 文件，结果如图 11-12 所示。

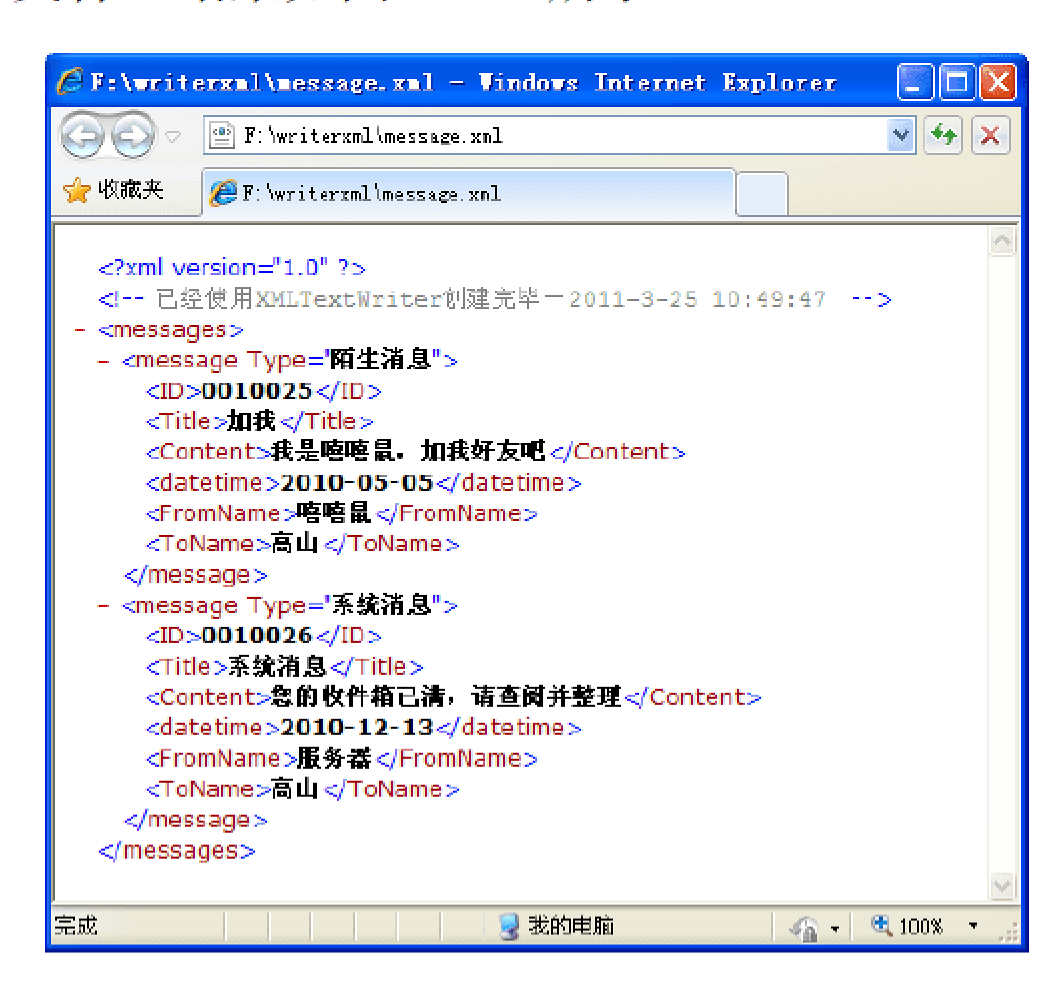
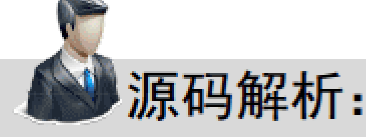


图 11-12　XML 文件

11.2.5　实例分析



使用文本创建 XML 文件时，添加的数据比较多时，要记得关闭元素。我们可以使用 XmlTextReader 类读取 XML 文件，根据其返回的结果是否为空判断该 XML 文件是否存在。

11.3　宠物信息的增删改操作

上面的两个章节我们主要讲了如何读取和写入 XML 文件，本章节中我们将讲解如何对 XML 文件进行增删改的操作。

接下来，我们以“宠物信息”为例，讲解如何对 XML 文件进行增加、删除、修改的操作。



视频教学：光盘/videos/11/宠物信息的增删改操作.avi　　长度：10 分钟

11.3.1　实例描述

如果说读取和写入 XML 文件时，针对的是 XML 文档的整体的处理。那么本节所讲的则是针对 XML 文档中属性或节点的增加、删除和修改操作，并将最终结果显示在页面上。

有时，我们将 XML 文件读取并显示到了页面上时，会发现需要添加一条数据或者修改某条数据，那么在不直接对 XML 文件进行操作的前提下，要如何进行操作呢，下面就让我们来看看吧。

11.3.2　实例应用

【例 11-3】　宠物信息的增删改操作

(1)　新建一个 ASP.NET 项目，添加 XML 文件 pet.xml，下面是 XML 文件内容：

```
<?xml version="1.0" encoding="utf-8"?>
<petlist>
  <pet master="Allen">
    <petname>小乖</petname>
    <age>3</age>
    <pettype>拉布拉多犬</pettype>
    <birth>加拿大</birth>
  </pet>
  <pet master="Lily">
    <petname>咪咪</petname>
    <age>2</age>
    <pettype>波斯猫</pettype>
    <birth>土耳其</birth>
  </pet>
</petlist>
```

(2)　添加 Default.aspx 页面，因为无须对前台页面进行布局，所以在 Default.aspx 后台页面 Default.aspx.cs 中添加如下代码，将 XML 文件内容显示在页面上：

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;
using System.Data;
public partial class  Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        ShowXmlValue();
    }
    /// <summary>
    /// 显示 XML 信息
    /// </summary>
    private void ShowXmlValue()
    {
        //显示
        string XMLFile = Server.MapPath("pet.xml");
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.Load(XMLFile);
        XmlNode xn = xmlDoc.SelectSingleNode("petlist");
        XmlNodeList xnl = xn.ChildNodes;
        foreach (XmlNode xnf in xnl)
        {
            XmlElement pet = (XmlElement)xnf;
            Response.Write("主人名称: " + pet.GetAttribute("master") + "<br>");
                                                    //显示属性值
            XmlNodeList xnf1 = pet.ChildNodes;
            foreach (XmlNode xn2 in xnf1)
            {

                                                    //显示子节点文本
                Response.Write("宠物信息: " + xn2.InnerText + "<br>");
            }
            Response.Write("<hr/>");
        }
    }
}
```

在该例中我们使用 DOM 技术读取 XML 文件，这样能更好地控制单个节点和属性的显示。首先，使用 XmlNode 获得 XML 文档的第一个节点，然后使用 XmlNodeList 集合该节点下的所有子节点，最后使用 foreach 循环显示。XmlElement 表示一个元素，GetAttribute 是指该元素的属性，ChildNodes 指该元素下的所有子节点。

(3) 添加方法 AddXMLValue，该方法表示在<petlist>节点中插入 master 属性值为“小可”的<pet>节点，代码如下：

```
/// <summary>
```

```
/// 增加节点
/// </summary>
private void AddXMLValue()
{

    string XMLFile = Server.MapPath("pet.xml");
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(XMLFile);
    XmlNode root = xmlDoc.SelectSingleNode("petlist"); //查找<petlist>
    XmlElement pet = xmlDoc.CreateElement("pet");      //创建一个<pet>节点
    XmlElement petname = xmlDoc.CreateElement("petname");
    petname.InnerText = "叽叽";                          //设置文本节点
    pet.AppendChild(petname);                          //添加到<pet>节点中
    pet.SetAttribute("master", "小可");
    XmlElement age = xmlDoc.CreateElement("age");
    age.InnerText = "1";
    pet.AppendChild(age);
    XmlElement pettype = xmlDoc.CreateElement("pettype");
    pettype.InnerText = "黄金鼠仓鼠";
    pet.AppendChild(pettype);
    XmlElement birth = xmlDoc.CreateElement("birth");
    birth.InnerText = "叙利亚";
    pet.AppendChild(birth);
    root.AppendChild(pet);//添加到<petlist>节点中
    xmlDoc.Save(XMLFile);
}
```

从代码中我们可以看到，添加操作和写入 XML 文档十分相似，不同的是，添加操作不用重新创建 XML 文档。

(4) 添加方法 UpdXMLValue，它表示对新添加的节点进行修改，将 master 属性值改为“王小可”，将子节点<petname>的文本改为“小波”，代码如下：

```
/// <summary>
/// 修改
/// </summary>
private void UpdXMLValue()
{
    string XMLFile = Server.MapPath("pet.xml");
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(XMLFile);
    XmlNodeList nodeList = xmlDoc.SelectSingleNode("petlist").ChildNodes;
                                       //获取 petlist 节点的所有子节点
     foreach (XmlNode xn in nodeList)  //遍历所有子节点
    {
        XmlElement pet = (XmlElement)xn;//将子节点类型转换为 XmlElement 类型
        if (pet.GetAttribute("master") == "小可")
        {
            pet.SetAttribute("master","王小可");  //修改属性值
            XmlNodeList nls = pet.ChildNodes;    //继续获取 pet 子节点的所有子节点
```

```
                foreach (XmlNode xn1 in nls)                    //遍历
                {
                    XmlElement petChild = (XmlElement)xn1;      //转换类型
                    if (petChild.Name == "petname")             //如果找到
                    {
                        if (petChild.InnerText == "叽叽")         //修改
                        {
                            petChild.InnerText = "小波";
                        }
                        break;                                  //修改后跳出循环
                    }
                }
            }
        }
        xmlDoc.Save(XMLFile);                                   //保存
}
```

在该方法中，遍历子节点，先将所有子节点类型转换为 XmlElement 类型，然后使用 GetAttribute 找到指定名称的属性值，接着使用 SetAttribute 修改属性值。修改子节点内容也一样。

(5) 添加方法 DelXMLValue，它表示删除“<pet master="黛玉">”节点的 master 属性，同时删除“<pet master="王小可">”节点。代码如下。

```
/// <summary>
/// 删除
/// </summary>
private void DelXMLValue()
{
    string XMLFile = Server.MapPath("pet.xml");
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(XMLFile);
    XmlNodeList xnl = xmlDoc.SelectSingleNode("petlist").ChildNodes;
    foreach (XmlNode xn in xnl)
    {
        XmlElement pet = (XmlElement)xn;
        if (pet.GetAttribute("master") == "黛玉")
        {
            pet.RemoveAttribute("master");              //移除属性
        }
        else if (pet.GetAttribute("master") == "王小可")
        {
            pet.RemoveAll();                            //删除该节点
            pet.ParentNode.RemoveChild(pet);            //删除其本身
        }
    }
    xmlDoc.Save(XMLFile);
}
```

11.3.3　运行结果

我们分别在 Default.aspx 页面的 load 事件里引用定义的三个方法，然后查看其运行效果。原始代码的运行效果如图 11-13 所示。

添加 AddXMLValue 方法后运行该页面，运行结果如图 11-14 所示。

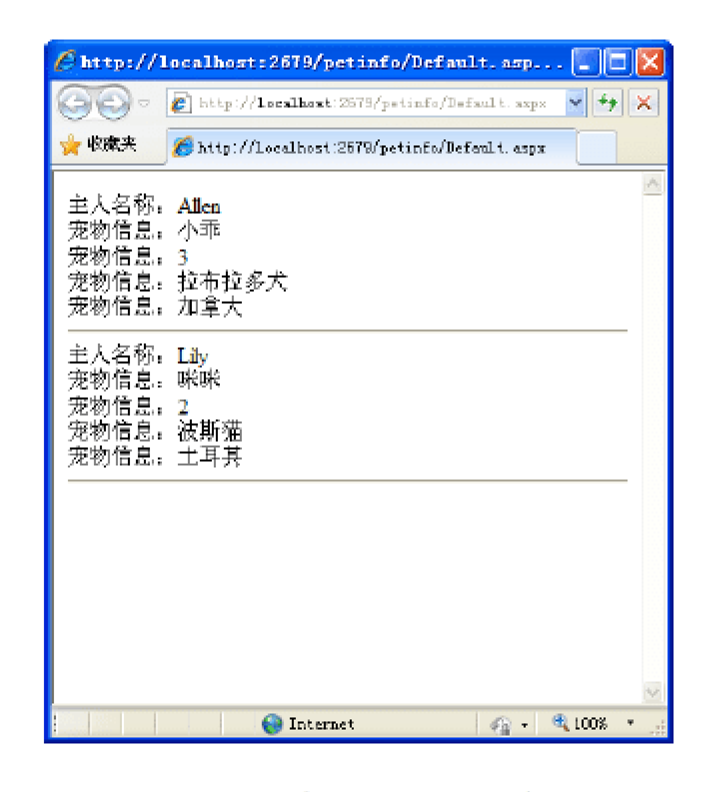


图 11-13　未改动的运行结果

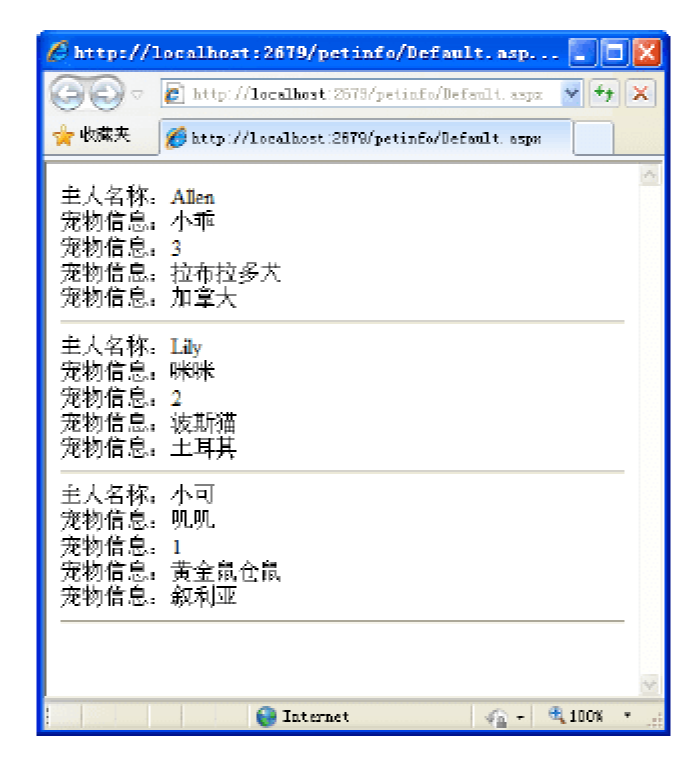


图 11-14　添加方法后的运行结果

添加 UpdXMLValue 方法后的运行结果如图 11-15 所示。

添加 DelXMLValue()方法后的运行结果如图 11-16 所示。

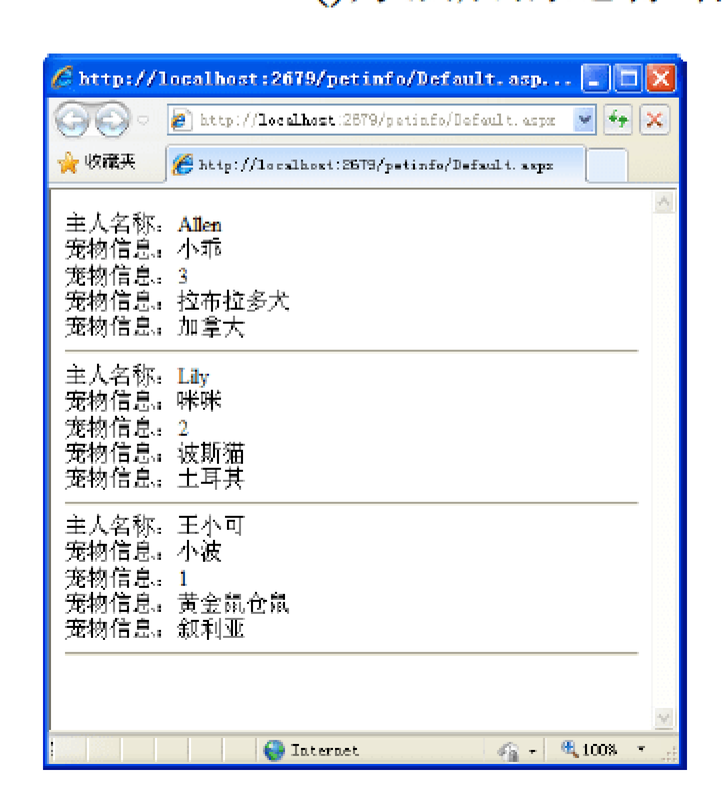


图 11-15　修改后的结果

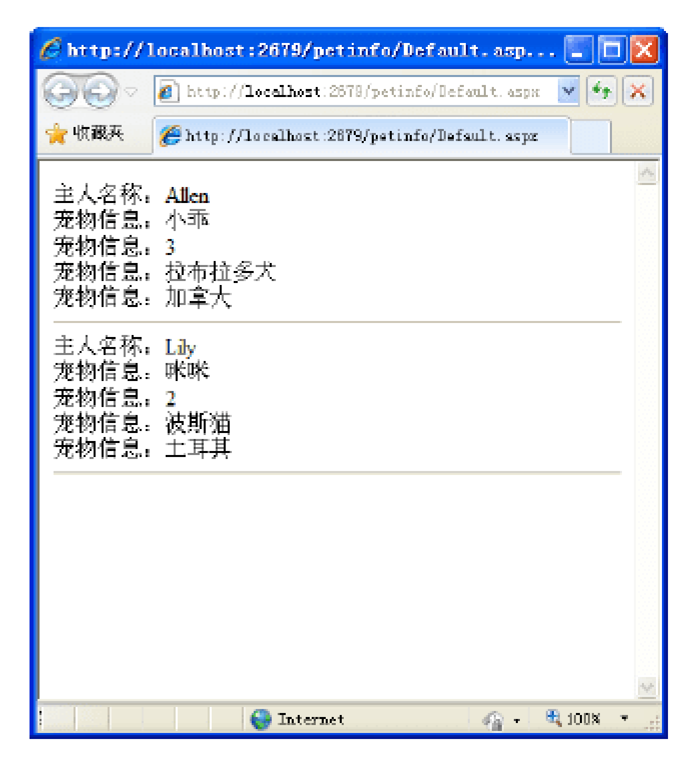


图 11-16　删除后的结果

11.3.4　实例分析



源码解析：

在删除 DelXMLValue 方法中，我们使用 RemoveAttribute 来移除属性值，还使用 RemoveAll 来移除该节点，RemoveAll 表示移除当前节点的所有属性和子节点，不移除默认属性。在使用 RemoveAll 时，必须要指定其属性值，否则无效。

11.4 自定义 XML 序列化

序列化就是将对象转换成易于传输形式的过程。为了方便存储和传输数据，我们将一个对象的公共域和属性保存为 XML 格式的过程就被称为 XML 序列化，也可称为 XML 串行化。

**视频教学**：光盘/videos/11/自定义 XML 序列化.avi　　长度：10 分钟

.NET Framework 中的 XmlSerializer 类为我们提供了 XML 序列化的功能。XmlSerializer 类位于 System.Xml.Serialization 命名空间下，它可以将一个对象串行化为 XML 格式。下面就详细讲解 XmlSerializer 类如何进行 XML 序列化。

1. XmlSerializer

XML 序列化的核心是 XmlSerializer 类，该类最重要的两个方法是 Serialize()和 Deserialize()方法。Serialize 方法可以实现 XML 序列化，Deserialize 方法可以实现 XML 反序列化。

要想对一个对象进行序列化，可以在它的前面加上[Serializable()]属性，它适用整个类，表示整个类都支持序列化。另外，还有[NonSerializabled()]属性，它适用于标有[Serializable()]的类中的字段，如果要被序列化的类中有某个字段不想被序列化，只需要在该字段前添加[NonSerializabled()]属性即可。除了这两个属性外，还有一些属性控制 XmlSerializer 类进行 XML 序列化，如表 11-3 所示。

表 11-3　控制 XmlSerializer 执行的属性

属　性	说　明
[XmlRoot]	用于识别作为 XML 文件根元素的类或者结构，可以用它把一个元素名设置为根元素
[XmlElement]	当公有的属性或者字段可以作为一个元素被串行化到 XML 结构中时使用
[XmlAttribute]	当公有的属性或者字段可以作为一个属性被串行化到 XML 结构中时使用
[XmlIgnore]	当公有的属性或者字段不包括在串行化的 XML 结构中时使用
[XmlArray]	当公有的属性或者字段可以作为一个元素数组被串行化到 XML 结构中时使用
[XmlArrayItem]	用于识别可以放在一个串行化数组中的类型

2. 基本序列化

序列化又分为基本序列化和定制序列化。

基本序列化是指.NET Framework 自动地对对象进行序列化操作。只需要相应的对象拥有[Serializable()]属性就可以了。使用这种方法，我们可以通过[NonSerializabled()]属性对每个字段进行精确控制，不想被序列化的字段只需要添加[NonSerializabled()]属性即可。例如下面所示的代码：

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Web;
using System.Xml.Serialization;
/// <summary>
///Books 的摘要说明
/// </summary>
[Serializable()]
[XmlRoot("Books")]
public class Books
{
    public Books() { }
    [XmlElement("BookName")]
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    [XmlElement("Author")]
    private string author;
    public string Author
    {
        get { return author; }
        set { author = value; }
    }
    [NonSerialized]
    [XmlElement("bookContent")]
    private string content;
    public string Content
    {
        get { return content; }
        set { content = value; }
    }
}
```

这段代码序列化后显示的结果如下：

```
<Books>
<BookName>返回的值</BookName>
<Author>返回的值</Author>
</Books>
```

因为 content 字段加了[NonSerializabled()]属性，所以并没有输出。



XML 序列化只能序列化公共类、公共字段和公共属性。

3．定制序列化

很明显，使用基本序列化自动对对象进行序列化操作，并不能很灵活地进行控制。所以我

们要使用定制序列化，这样就能够精确地确定哪个字段需要序列化，以及怎样进行序列化。

使用定制序列化，首先需要添加[Serializable()]属性，然后还需要实现 ISerializable 接口，该接口声明如下所示：

```
public interface ISerializable
{
    void GetObjectData(SerializationInfo info, StreamingContext context);
}
```

该接口位于 System.Runtime.Serialization 命名空间下。ISerializable 接口只有一个 GetObjectData()方法，该方法的主要作用是为了将目标对象序列化所需的数据填充到 SerializationInfo。

例如，我们可以通过下列代码来了解怎样定制序列化。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Runtime.Serialization;
/// <summary>
///Students 的摘要说明
/// </summary>
[Serializable()]
public class Student : ISerializable
{
    public string name = null;
    public int age = 0;
    public bool sex = true;
    public Student()
    {
    }
    protected Student(SerializationInfo info, StreamingContext context)
    {
        name = info.GetString("name");
        age = info.GetInt32("age");
        sex = info.GetBoolean("sex");
    }
    void ISerializable.GetObjectData(SerializationInfo info, StreamingContext
      context)
    {
        info.AddValue("name", name);
        info.AddValue("age", age);
        info.AddValue("sex", sex);
    }
}
```

可以看到，本例中所声明的三个字段都是公共的，且具有不同类型。但是在创建的 GetObjectData()方法中只需直接调用 AddValue()方法即可，因为 AddValue()方法带有所有标准类型的重载方法。

11.5 常见问题解答

11.5.1 关于 XML 文件写入的问题



关于 XML 文件写入的问题。
网络课堂：http://bbs.itzcn.com/thread-15441-1-1.html

我有一个 XML 文件，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration  xmlns="urn:nhibernate-configuration-2.2" >
  <session-factory name="SeNet.Base.NHibernate">
    <!-- properties -->
    <property name="connection.connection_string">0123456</property>
       <!-- mapping files -->
    <!-- are now optional
   <mapping file="ABC.hbm.xml" />
   <mapping resource="NHibernate.DomainModel.Simple.hbm.xml"
assembly="SeNet.BPZS.Entity" />
  -->
    <mapping assembly="SeNet.BPZS.Entity" />
  </session-factory>
</hibernate-configuration>
```

如果我想修改里面的<property name="connection.connection_string">0123456</property>为<property name="connection.connection_string">789123</property>，我应该怎么做？

如果我按照以下方式修改：

```
string retXml =
"<property>name='connection.connection_string'>0123456</property>";
                                              //此处省略
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(retXml);
    XmlNodeList nodes = doc.GetElementsByTagName("property ");
    nodes[0].InnerText = "789123";
    doc.Save("XMLFile1.xml");
```

则会显示“未将对象引用设置到对象的实例”的错误。若按照以下方式修改：

```
    XmlDocument doc = new XmlDocument();
    doc.LoadXml("XMLFile1.xml");
    XmlNodeList nodes = doc.GetElementsByTagName("property ");
    nodes[0].InnerText = "789123";
    doc.Save("XMLFile1.xml");
```

这样修改之后，显示的错误为“根级别上的数据无效”。

怎么办？

【解决办法】

可以将代码改写如下：

```
string
retXml="<property>name='connection.connection_string'>0123456</property>";
//此处省略
XmlDocument doc = new XmlDocument();
  doc.LoadXml(retXml);
  XmlNodeList   nodes   =   doc.GetElementsByTagName("property ");
  nodes[0].InnerText   =   "789123";
  doc.Save("XXX.xml");
```

11.5.2 Google 地图 XML 的写入问题



Google 地图 XML 的写入问题。

网络课堂：http://bbs.itzcn.com/thread-15442-1-1.html

```
XmlDocument doc = new XmlDocument();
doc.Load(Server.MapPath("~/GoogleMap article.xml"));
XmlNode n = doc.SelectSingleNode("/urlset/url/loc");
n.InnerXml = this.TextBox1.Text;
doc.Save(Server.MapPath("~/GoogleMap_article.xml"));
```

此段代码要写入 GoogleMap_article.xml 文件。

这个文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.google.com/schemas/sitemap/0.84">
<url>
<loc>http://www.tjanju.cn</loc>
</url>
</urlset>
```

可是问题来了，写不进去。只有改成：

```
<urlset>
<url>
<loc>http://www.tjanju.cn</loc>
</url>
</urlset>
```

才可以写进去，可是写进去了吧，又不能追加，我要的是每次追加一行。

```
<url>
<loc>http://www.tjanju.cn</loc>
</url>
```

【解决办法】

这个很简单。我直接给你实现了，看看下面的代码吧：

```
XmlDocument doc = new XmlDocument();
doc.Load(Server.MapPath("~/GoogleMap_article.xml"));
XmlNode node = doc.DocumentElement.ChildNodes[0].Clone();
XmlNode n = doc.SelectSingleNode("/urlset/url/loc");
n.InnerText = TextBox1.Text;
doc.DocumentElement.AppendChild(node);
doc.Save(Server.MapPath("~/GoogleMap_article.xml"));
```

11.5.3 在 XML 指定位置写入



在 XML 指定位置写入。

网络课堂：http://bbs.itzcn.com/thread-15443-1-1.html

我有一段代码如下：

```
XmlDocument xd = new XmlDocument();
xd.Load(AppDomain.CurrentDomain.BaseDirectory+"web.config");
XPathNavigator xn = xd.CreateNavigator();
XPathNodeIterator xni = xn.Select("configuration/connectionStrings");
XmlNode x = ((IHasXmlNode)xni.Current).GetNode();
```

再往下不会写了。

我想在 web.config 中的“<connectionStrings></connectionStrings>”里写入下面内容：

```
<add name="connString"
connectionString="server=.;database=tempdb;uid=sa;pwd=sa"/>
```

希望高手们能把你们的解决方案贴在下面。

【解决办法】

我以前公司研究了几个月的 XML，基本所有格式都能写出来。不懂得都可以问我，在节点里加下级节点：

```
XmlNode Add= xd.CreateNode("element", "add", "");
xni.AppendChild(Add);
CreateAttribute(Add, "name", "connString");
CreateAttribute(Add, "connectionString",
"server=.;database=tempdb;uid=sa;pwd=sa");
```

往节点加属性：

```
CreateAttribute(xni, "name", "张小华");
public XmlAttribute CreateAttribute(XmlNode node, string attributeName, string
value)
{
    try
    {
```

```
        XmlDocument doc = node.OwnerDocument;
        XmlAttribute attr = null;
        attr = doc.CreateAttribute(attributeName);
        attr.Value = value;
        node.Attributes.SetNamedItem(attr);
        return attr;
    }
    catch (Exception err)
    {
        string desc = err.Message;
        return null;
    }
}
```

11.6 习　　题

一、填空题

(1) 读取 XML 文档的 4 种方式分别是__________、使用 XML 控件、使用 DOM 技术和使用 DataSet 方式读取。

(2) 使用 XML 控件读取 XML 文档时要将 XML 文档的路径赋给 XML 控件的__________属性。

(3) 使用定制序列化必须要实现的接口是__________。

(4) 如果在要序列化的某个类中，有一个字段不需要序列化，只需要在该字段前添加__________属性即可。

二、选择题

(1) 下列关于 XML 特性的描述中，不正确的一项是__________。

A. XML 是纯文本格式，与平台无关

B. XML 是可扩展的、自定义的文档，有利于不同系统定义不同的标准文档

C. XML 将文档的数据、结构和实现方式结合在一起

D. XML 的数据存储方式不受显示格式的制约

(2) 使用 XmlTextWriter 方式创建 XML 文档时，如果想为节点添加一个属性，我们可以使用 XmlTextWriter 的__________方法。

A. WriteAttributeString

B. WriteElementString

C. WriteComment

D. WriteStartAttribute

(3) 下列__________不属于 XMLReader 的三个派生类。

A. XmlNodeReader

B. XmlValueReader

C. XmlValidatingReader

D. XmlTextReader

(4) 下列关于 XML 序列化的描述正确的一项是__________。

A. XML 序列化只能序列化公共类、公共字段和公共属性

B. 使用基本序列化可以完全控制序列化和反序列化的行为

C. 定制序列化可以不用实现 ISerializable 接口

D. 基本序列化也能实现 ISerializable 接口

三、上机练习

上机练习 1： 实现宠物信息展示，以及动态添加宠物信息。

以 11.3 节的实例为基础，实现在页面显示宠物信息，如图 11-17 所示。单击【添加信息】链接到 AddPet.aspx 页面，实现动态添加信息到 XML 文档，如图 11-18 所示。单击【查看添加】按钮回到 Default.aspx 页面，查看新添加的信息。



图 11-17 显示宠物信息

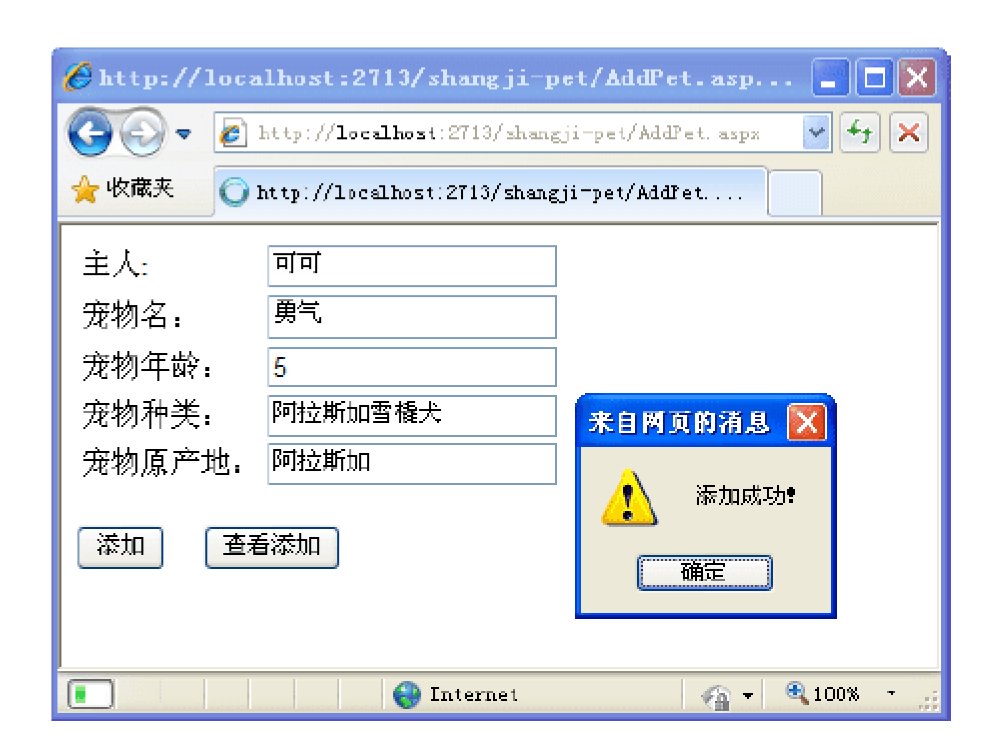


图 11-18 添加宠物信息



第 12 章　集成第三方 Web 服务

内容摘要：

在本章之前我们学习了如何开发一个 Web 服务。事实上，在实际开发中，有很多 Web 服务并不需要我们亲自动手编写。比如天气查询，我们的网站需要展示和查询天气情况，如果靠我们自己去编写，会非常麻烦，怎么解决呢？这时就用到第三方 Web 服务了，在我们的网站中只需要添加 Web 服务的引用就可以了。

学习目标：

- 熟练使用第三方 Web 服务实现验证码功能
- 熟练使用第三方 Web 服务查询手机号码归属地
- 熟练使用第三方 Web 服务查询 IP 地址所在地
- 熟练使用第三方 Web 服务查询邮政编码
- 熟悉掌握第三方 Web 服务查询火车班次
- 熟悉掌握第三方 Web 服务查询各个城市天气状况

12.1　实现后台登录时的验证码

我们在逛论坛或社区时，经常会看到登录或发帖时需要输入验证码的文本框，使用验证码是防止网站被恶意攻击的有效手段之一。可是，如果不会编写验证码该怎么办呢？别急，我们可以在网上找到第三方提供的验证码 Web 服务，将它引入我们的项目就可以了，下面就让我们来学习如何使用第三方提供的 Web 服务吧。


视频教学：光盘/videos/12/实现后台登录的验证码.avi　　长度：11 分钟

12.1.1　实例描述

最近在写一个后台登录系统，为了防止被恶意登录或暴力破解，决定添加一个验证码，可是不会写验证码，只好在网上找到一个提供验证码的第三方 Web 服务。下面就来和大家一起分享如何使用第三方 Web 服务。

12.1.2　实例应用

【例 12-1】 实现后台登录时的验证码

(1) 打开 Microsoft Visual Studio 2010，新建一个 ASP.NET 项目，并为项目添加 Web 引用。第三方 Web 服务的 URL 为“http://webservice.webxml.com.cn/WebServices/ValidateCodeWebService.asmx”，修改 Web 引用名为 ValidateCodeWS，如图 12-1 所示。

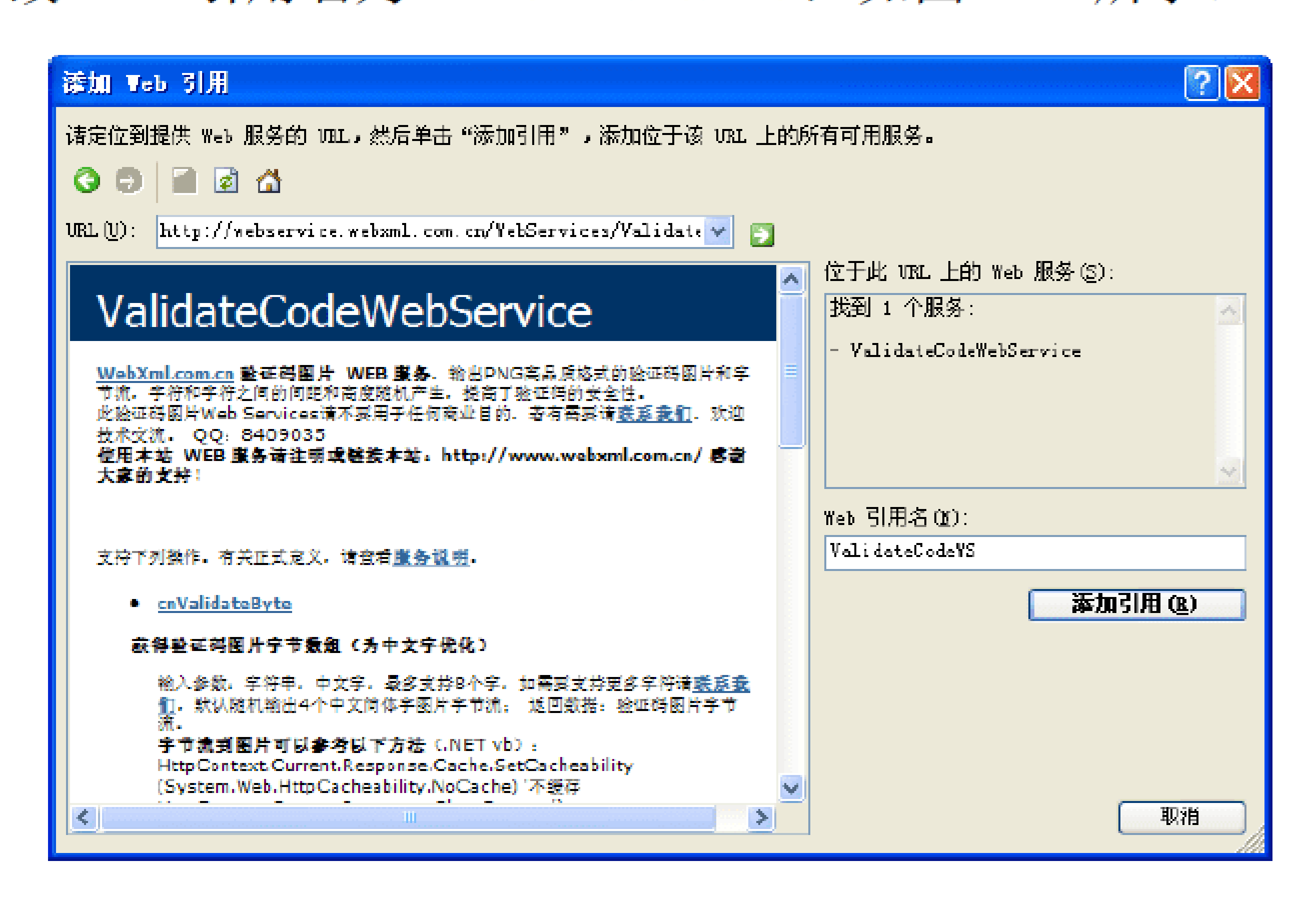


图 12-1　添加 Web 引用

(2) 在项目中添加页面 Login.aspx 和 ValidateCode.aspx。ValidateCode.aspx 页面用于生成验证码，在 ValidateCode.aspx.cs 页面中添加如下代码：

```
public partial class ValidateCode : System.Web.UI.Page
{
```

```
    ValidateCodeWS.ValidateCodeWebService vcws = new
ValidateCodeWS.ValidateCodeWebService();            //实例化 Web 服务对象
    protected void Page Load(object sender, EventArgs e)
    {
        if(Request.Params["validateDate"]!=null)//接收传来的参数
        {
            int i=Int32.Parse( Request.Params["validateDate"].ToString());
            Response.ContentType = "image/Png";
            byte[] b = vcws.enValidateByte(i.ToString());
            Response.BinaryWrite(b);
        }
    }
}
```

“Request.Params["validateDate"]”表示在 Login.aspx 页面里随机生成的对象，将它作为参数传给 ValidateCode.aspx 页面。在 ValidateCode.aspx 页面里实例化 Web 服务对象，以“Request.Params["validateDate"]”为参数，返回一个二进制字符串，并且将它写入 HTTP 输出流。

(3) 在 login.aspx.cs 页面，添加如下代码，实现登录功能和验证码的生产：

```
public partial class ValidatePicture : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            Random rd = new Random();                    //实例化随机对象
            string str = rd.Next(1000, 9999).ToString();
            Image1.ImageUrl = "ValidateCode.aspx?validateDate=" + str;
                                                         //生成随机验证码图片
            Session["validateDate"] = str;
        }
    }
    protected void Image1_Click(object sender, ImageClickEventArgs e)
                                                         //换一张图片
    {
        Random rd = new Random();                        //实例化随机对象
        string str = rd.Next(1000, 9999).ToString();
        Image1.ImageUrl = "ValidateCode.aspx?validateDate=" + str;
                                                         //生成随机验证码图片
        Session["validateDate"] = str;
    }
    protected void btnOK_Click(object sender, EventArgs e)
    {
        if (txtUser_name.Value.Trim() == "admin" && txtUser_pwd.Value.Trim() ==
            "admin")
        {
            if (Session["validateDate"].ToString() ==
                txtValidateCode.Value.Trim())
```

```
            {
                ClientScriptManager csm = Page.ClientScript;
                csm.RegisterStartupScript(GetType(), "", "<script>alert
                    ('登录成功！');</script>");
            }
            else
            {
                ClientScriptManager csm = Page.ClientScript;
                csm.RegisterStartupScript(GetType(), "", "<script>alert
                    ('登录失败！');</script>");
            }
        }
    }
}
```

首先在 login.aspx.cs 页面的 Load 事件里，实例化随机对象，并且生成一个 1000～9999 之间的随机数，将随机数作为参数传递给 ValidateCode.aspx 页面，并且将它保存到 Session 里。同时，为了实现点击图片换一张图片的功能，我们将验证码放在一个 ImageButton 控件中，并且在 ImageButton 控件的单击事件里添加和 Load 事件同样的代码。最后，在 btonOK 登录按钮的 Click 事件里，验证输入的验证码和生成的验证码是否相同，即将 txtValidateCode 文本框里输入的验证码和 Session 里保存的验证码进行比较。

12.1.3　运行结果

运行 Login.aspx 页面，在浏览器中打开该页面，可以看到带有验证码的登录页面，如图 12-2 所示。



图 12-2　有验证码的登录页面

单击验证码图片，可更换验证码。输入用户名称“admin”和用户密码“admin”，将生成的验证码输入到文本框中，单击【登录】按钮，验证成功，显示登录成功对话框，运行结果如图 12-3 所示。



图 12-3　验证成功的登录页面

12.1.4　实例分析



源码解析：

本实例中的 ValidateCode.aspx 页面负责接收传递过来的数据，并调用第三方 Web 服务提供的接口 enValidateByte()，返回的验证码图片字节流被 Login.aspx 页面的 ImageButton 控件接收，然后显示在界面上。本例使用的第三方 Web 服务还提供了很多接口来实现验证码的功能，读者可以试一下。

12.2　手机号码归属地查询

前面我们对第三方 Web 服务的应用有了一个简单的了解，它不但能减少网站开发人员的工作量，还能给我们的生活带来便利。下面我们一起去学习使用手机号码归属地查询的 Web 服务。

 **视频教学：**光盘/videos/12/手机号码归属地查询.avi　　长度：5 分钟

12.2.1　实例描述

有一段时间，经常接到一个陌生的电话，接通后对方却一直不说话，因为害怕是公司的客户，所以再打来的时候又不敢不接。于是自己以前写的一个手机号码归属地查询有了用武之地，查询后才知道，这个号码是新疆的，公司在新疆根本没业务啊，赶紧把这个号码设置成拒接号码。

学到的东西总算是用到了日常生活中，感觉很有成就感，下面我们一起去看看这个实例，学会后也能向同学和朋友炫耀一下。

12.2.2 实例应用

【例 12-2】 手机号码归属地查询

(1) 打开上节创建的 ASP.NET 项目，首先添加本节所需的第三方 Web 服务引用，Web 服务的 URL 为“http://webservice.webxml.com.cn/WebServices/MobileCodeWS.asmx”，Web 引用名为“mobileService”。

(2) 添加 TelephoneSearch.aspx 页面，前台部分代码如下所示：

```
<table width="600" border="0" align="center" cellpadding="1" cellspacing="1"
bgcolor="#a9cfe4">
<tr>
   <td height="32" align="center" bgcolor="#a9cfe4" class="title1">
       手机号码归属地查询
   </td>
</tr>
<tr>
   <td  height="45" align="center" bgcolor="#FFFFFF" valign="middle">
  手机号码(最少前七位数字)：<asp:TextBox ID="txtMobileNum" runat="server"
MaxLength="11"></asp:TextBox>
  <asp:Button ID="btnSearch" runat="server" OnClick="btnSearch_Click" Text="
查询" Width="49px" />
          <br />
  <asp:UpdateProgress ID="UpdateProgress1" runat="server">
<ProgressTemplate>
   <img src="images/loading.gif" alt="Loading... ..." />
    </ProgressTemplate>
 </asp:UpdateProgress>
    <asp:Label ID="lblShow" runat="server"></asp:Label>
   </td>
  </tr>
</table>
```

因为涉及到 Web 访问，有时候在查询结果出来前需要很长时间，为了防止页面长时间没有响应，在这里我们利用 Ajax 技术，放上一个进度条，使页面在结果出来前有一个缓冲，在这里涉及到的 Ajax 技术不再详细讲解。

(3) 在 btnSearch 按钮的 Click 事件里添加如下代码：

```
protected void btnSearch_Click(object sender, EventArgs e)
{
  mobileService.MobileCodeWS mc = new mobileService.MobileCodeWS();
                                                   //实例化 Web 服务对象
   string showMsg = mc.getMobileCodeInfo(txtMobileNum.Text.Trim(), "");
                                                   //查询输入手机号码的归属地
   lblShow.Text = "手机号码：" + txtMobileNum.Text.Trim() + "<br/>" + "归属地"
    + showMsg.Substring(txtMobileNum.Text.Length);
}
```

12.2.3　运行结果

运行该页面，输入你要查询的手机号码，结果如图 12-4 所示。



图 12-4　手机号码归属地查询结果

12.2.4　实例分析

**源码解析：**

本实例中调用了第三方 Web 服务提供的 getMobileCodeInfo()方法，只要手机号码(最少前 7 位)作为参数，便能查询出手机的归属地。方法的参数和返回类型在提供此服务的 Web 服务上有详细描述。输入参数：mobileCode =字符串(手机号码，最少前 7 位数字)，userID=字符串(商业用户 ID)，userID 可以为空；返回数据：字符串(手机号码：省份 城市 手机卡类型)。

12.3　IP 地址查询

通过上节课的学习，相信大家对第三方 Web 服务有了更深入的了解，也会更有兴趣去学习第三方 Web 服务，因为它能使我们有成就感。既然有了学习兴趣，咱们就赶紧开始这节课的学习吧。

**视频教学：**光盘/videos/12/IP 地址查询.avi　　　　长度：5 分钟

12.3.1　实例描述

刚开始接触互联网的时候，浏览一些门户网站，上面总是显示着我所在的城市，而且后面跟着就是天气状况，很好奇那些门户网站怎么知道我们这些浏览者在哪个城市，他们就不怕把我们的地址弄错吗？后来学习 Web 服务才明白，我的担心实在是多余，要知道浏览者在哪个

城市，原来是如此的简单。

12.3.2 实例应用

【例 12-3】 IP 地址查询

(1) 打开 12.1 节创建的项目，添加查询 IP 地址所需的第三方 Web 服务引用，Web 服务的 URL 为“http://webservice.webxml.com.cn/WebServices/IpAddressSearchWebService.asmx”，Web 引用名为“IpAddressSearchWebService”。

(2) 添加名为 IPSearch.aspx 的页面，页面前台添加 txtIP 文本框控件，用于输入 IP 地址，btnSearch 按钮用于查询，ID 为 lblShow 的 Label 控件用于显示查询结果。

(3) 在 IPSearch.aspx 页面的 Load 事件里添加如下代码，显示本机 IP 地址所在地：

```
IpAddressSearchWebService.IpAddressSearchWebService ipWS
= new IpAddressSearchWebService.IpAddressSearchWebService();
protected void Page_Load(object sender, EventArgs e)
{
    string[] IPStr =ipWS.getGeoIPContext(); //获取本机 IP 信息
    lblShow.Text ="您当前的 IP 地址是："+ IPStr[0]+"来自："+IPStr[1]+"<br>";
}
```

(4) 在 btnSearch 按钮的 Click 事件里添加如下代码，查询输入到文本框中的 IP 的所在地：

```
protected void btnSearch_Click(object sender, EventArgs e)
{
    string[] IPStr =ipWS.getCountryCityByIp(txtIP.Text.Trim());
                                        //获取输入的 IP 地址的信息
    lblShow.Text += "您输入的 IP 地址是：" + IPStr[0] + "来自：" + IPStr[1];
}
```

12.3.3 运行结果

运行 IPSearch.aspx 页面，输入测试 IP，查询结果如图 12-5 所示。



图 12-5 IP 地址查询结果

12.3.4　实例分析



源码解析：

本实例调用了第三方 Web 服务提供的 getGeoIPContext()接口和 getCountryCityByIp()接口。getGeoIPContext()接口用来获取本机的 IP 地址和归属地，getCountryCityByIp()接口用来获取输入的 IP 地址的归属地。

把 Web 服务的 URL 输入到浏览器的地址栏中，可以查看此 Web 服务提供的方法的详细描述。

12.4　邮政编码查询

随着网络技术的飞速发展，我们越来越依赖网络带给我们生活的便利。可是，如果哪天我们必须要用到原始的交流方式寄信给对方时，就必须知道对方的邮政编码了，怎么能快捷地知道对方的邮政编码呢？在本节中，我们将利用第三方 Web 服务向大家展示如何快捷地查询对方的邮政编码。



视频教学：光盘/videos/12/邮政编码查询.avi　　　　长度：7 分钟

12.4.1　实例描述

我朋友在外地上学，本来平时联系的时候都是用聊天工具或者电子邮箱的，可是前段时间朋友学校全校禁网，没办法，只能使用原始的书信交流。可是，提起笔写完信后傻眼了，邮政编码该怎么填呀，我根本就不知道朋友那的邮政编码呀，哎，欲哭无泪呀。还好，某段时间做的一个小程序帮了我大忙，利用第三方 Web 服务查询邮政编码，这些小程序真是好用呀，下面就来看看怎么实现吧。

12.4.2　实例应用

【例 12-4】　邮政编码查询

(1)　打开 12.1 节创建的项目，添加第三方 Web 服务引用，Web 服务的 URL 为“http://webservice.webxml.com.cn/WebServices/ChinaZipSearchWebService.asmx”，Web 引用名为“ChinaZipSearchWebServices”。

(2)　添加名为 ZipSearch.aspx 的页面，页面前台布局如图 12-6 所示。

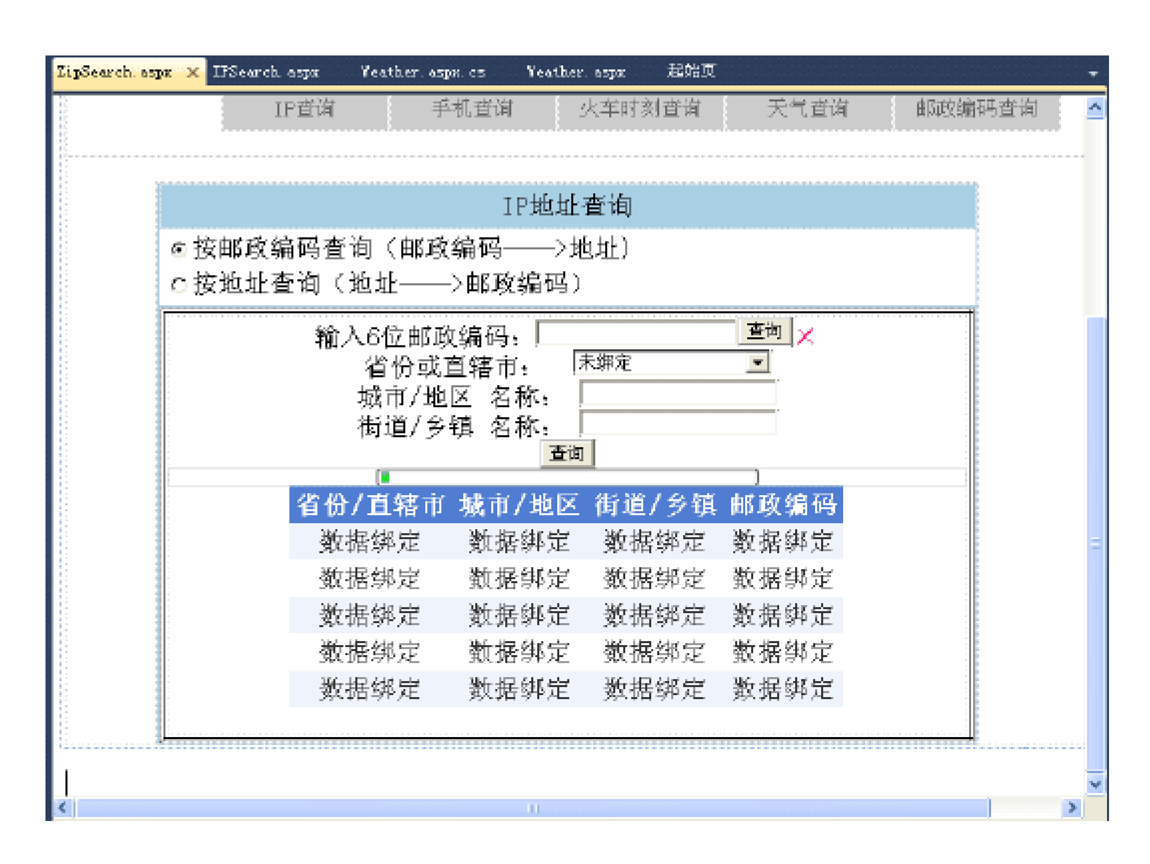


图 12-6　ZipSearch.aspx 页面前台布局

ZipSearch.aspx 页面的部分前台代码如下所示：

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server"
   OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged"
AutoPostBack="true">
  <asp:ListItem Value="1" Selected="True">按邮政编码查询(邮政编码——&gt;地址)
        </asp:ListItem>
  <asp:ListItem Value="2">按地址查询(地址——&gt;邮政编码)</asp:ListItem>
</asp:RadioButtonList>
<div id="ByZip" runat="server">
         输入 6 位邮政编码：<asp:TextBox ID="txtZip" runat="server"></asp:TextBox>
    <asp:Button ID="btnSearchByZip" runat="server" Text="查询"
OnClick="btnSearchByZip_Click" />
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
runat="server"ControlToValidate="txtZip" ErrorMessage="×"
ForeColor="#FF0066"
 ValidationExpression="\d{6}"></asp:RegularExpressionValidator>
        <br />
</div>
<div id="ByAddress" visible="false" runat="server">
      省份或直辖市： 
      <asp:DropDownList ID="Province" runat="server" Style="margin-left: 0px"
           Width="145px">
        </asp:DropDownList>
       <br />
      城市/地区 名称：
         <asp:TextBox ID="txtCity" runat="server" Width="145px"></asp:TextBox>
           <br />
      街道/乡镇 名称：
           <asp:TextBox ID="txtAddress" runat="server"
                 Width="145px"></asp:TextBox>
            <br />
             <asp:Button runat="server" ID="btnSearchByAddress" Text="查询"
                   OnClick="btnSearchByAddress Click" />
</div>
```

RadioButtonList 控件用于控制两个层 ByZip 和 ByAddress 的显示和隐藏。在 RadioButtonList 控件的 SelectedIndexChanged 事件里的代码如下：

```
protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (RadioButtonList1.SelectedValue == "1")
    {
        ByZip.Visible = true;
        ByAddress.Visible = false;
    }
    else if (RadioButtonList1.SelectedValue == "2")
    {
        ByZip.Visible = false;
        ByAddress.Visible = true;
    }
}
```

在该 Web 服务中，查询返回的数据类型是 DataSet 数据类型，所以还需要在前台页面添加用于显示查询所得的数据的数据控件，在这里我们使用的是 GridView 数据控件。

(3)　下面是页面 Load 事件的数据绑定，用于在页面加载时绑定省份的下拉列表框：

```
ChinaZipSearchWebServices.ChinaZipSearchWebService ChinaZip =
 new ChinaZipSearchWebServices.ChinaZipSearchWebService();
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        //绑定省份下拉列表框
        Province.DataSource = ChinaZip.getSupportProvince();
        Province.DataBind();
    }
}
```

(4)　接下来是按输入的邮政编码查询地址和按地址查询邮政编码的两种查询方法的实现代码：

```
protected void btnSearchByZip_Click(object sender, EventArgs e)
{
    //按输入的邮政编码查询地址
    DataSet ds = ChinaZip.getAddressByZipCode(txtZip.Text.Trim(), "");
    GridViewShowAddress.DataSource = ds.Tables[0].DefaultView;
    GridViewShowAddress.DataBind();
}
protected void btnSearchByAddress_Click(object sender, EventArgs e)
{
    DataSet ds =
ChinaZip.getZipCodeByAddress(Province.SelectedValue, txtCity.Text.Trim(),
txtAddress.Text.Trim(), "");
    //根据地址获取邮政编码
```

```
    GridViewShowAddress.DataSource = ds.Tables[0].DefaultView;
    GridViewShowAddress.DataBind();
}
```

12.4.3 运行结果

运行 ZipSearch.aspx 页面，按邮政编码查询如图 12-7 所示，按地址查询如图 12-8 所示。

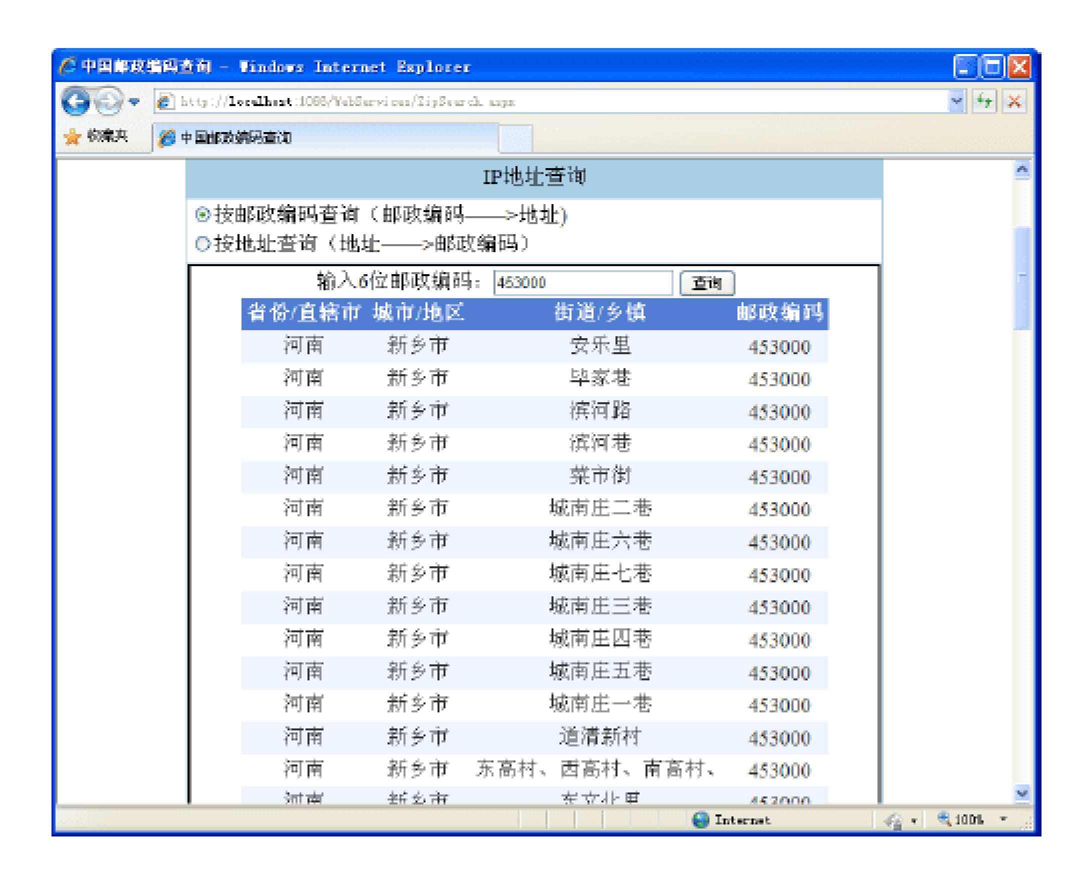


图 12-7 按邮政编码查询



图 12-8 按地址查询

12.4.4 实例分析



源码解析：

本实例中调用第三方 Web 服务提供的 getSupportProvince()接口，返回它支持的省份/城市信息，getAddressByZipCode()接口根据输入的邮政编码返回对应的地址，getZipCodeByAddress()接口根据输入的地址返回对应的邮政编码。使用 RadioButtonList 控件时，要想实现该控件的 OnSelectedIndexChanged 事件，必须先设定属性 AutoPostBack 为“true”，表示当选定内容更改时，自动回发到服务器上。

12.5　火车车次查询

出行是一个大问题，如果我们要坐火车的话，要是不知道坐哪次车，什么时候出发，什么时候到站，那可就糟糕了。所以，我们在出行前也必须要了解所有火车车次的信息，这样才不至于在临出发时手忙脚乱。

接下来，就教给大家一个简单的关于火车车次查询的程序，快点跟着我来学习吧。

**视频教学：**光盘/videos/12/火车车次查询.avi　　**长度：**5 分钟

12.5.1　实例描述

前段时间我朋友要出差去上海，不知道要坐哪趟火车去，刚好我在做一个关于查询火车车次的程序，利用了第三方 Web 服务，数据绝对准确，于是，借花献佛，也算是给朋友的一个离别礼物。下面就让我们一起来看看是如何实现的吧。

12.5.2　实例应用

【例 12-5】　火车车次查询

(1)　打开 12.1 节创建的 ASP.NET 项目，为项目添加本节所需的第三方 Web 服务引用，Web 服务 URL 为“http://webservice.webxml.com.cn/WebServices/TrainTimeWebService.asmx”，Web 引用名为“TrainTimeSearchWS”。

(2)　添加名为 TrainTimeSearch.aspx 的页面。前台页面布局如图 12-9 所示。代码略。



图 12-9　TrainTimeSearch.aspx 前台页面布局

两个 GridView 控件分别绑定火车车次和站站查询的结果。

(3)　查询按钮的单击事件代码如下所示：

```
protected void btnSearch_Click(object sender, EventArgs e)
{
    TrainTimeSearchWS.TrainTimeWebService ttws = new
TrainTimeSearchWS.TrainTimeWebService();
   if (TrainNum.Checked == true)//按火车车次查询
```

```
    {
        string[] str =ttws.getStationAndTimeByTrainCode
              (txtTrainNum.Value.Trim(),"");
        lblShow.Text = "车次: "+str[0]+", ";
        lblShow.Text += "始发站: " + str[3] + " , ";
        lblShow.Text += "发车时间: " + str[4] + " , ";
        lblShow.Text += "终点站: " + str[5] + " , <br/>";
        lblShow.Text += "到达时间: " + str[6] + ", ";
        lblShow.Text += "里程: " + str[7] + "KM, ";
        lblShow.Text += "历时时间: " + str[8];
        GridView1.DataSource =
          ttws.getDetailInfoByTrainCode(txtTrainNum.Value.Trim(), "").
          Tables[0].DefaultView;
        GridView1.DataBind();
        GridView2.Visible = false;
        GridView1.Visible = true;
        lblShow.Visible = true;
    }
    if (TrainZhanZhan.Checked == true)//按车站到车站查询
    {
         GridView2.DataSource =ttws.getStationAndTimeByStationName
             (txtChuFaZhan.Value.Trim(),txtDaoDaZhan.Value.Trim(),"");
         GridView2.DataBind();

          lblShow.Visible = false;
          GridView2.Visible = true;
          GridView1.Visible = false;
     }
}
```

12.5.3 运行结果

运行该页面，根据火车车次查询结果如图 12-10 所示。

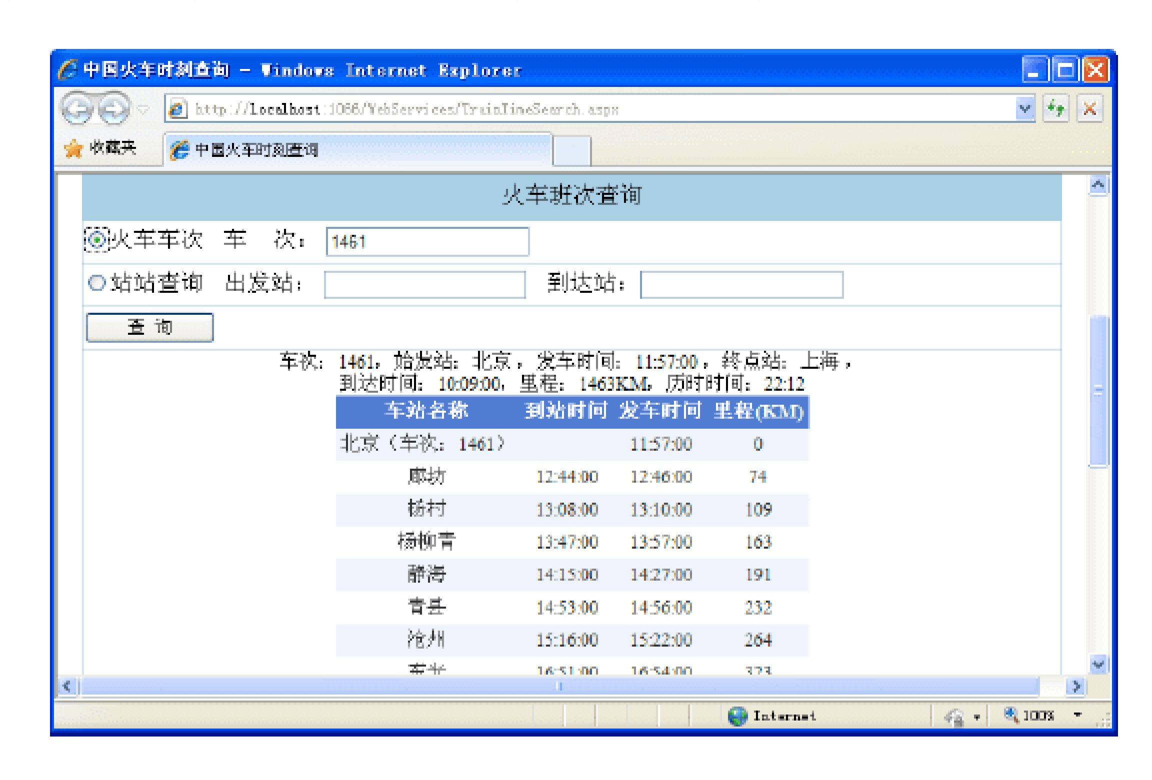


图 12-10 根据火车车次查询结果

在出发站和到达站分别输入城市名称，单击查询，查询结果如图 12-11 所示。

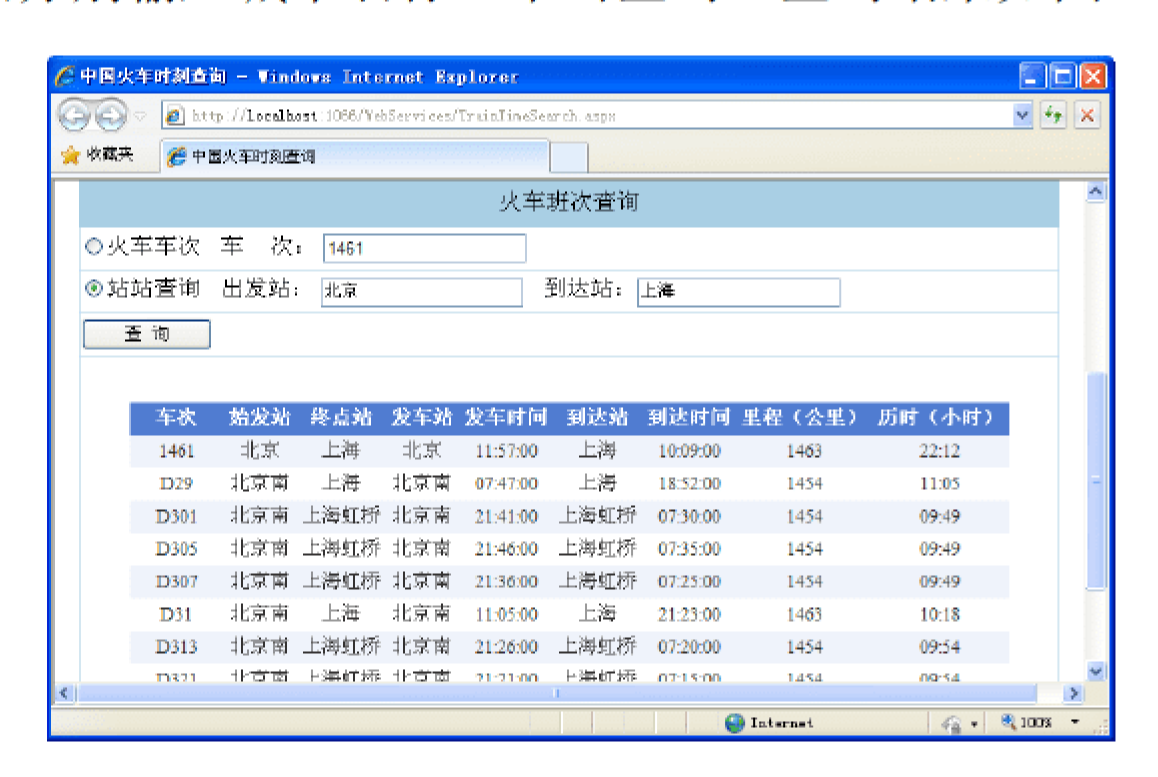


图 12-11　查询结果

12.5.4　实例分析

**源码解析：**

本例中，在查询按钮的单击事件里，我们根据 TrainNum 单选按钮和 TrainZhanZhan 单选按钮是否被选中的状态，来判断执行哪种查询方式。如果选择的是火车车次，则使用第三方 Web 服务提供的 getStationAndTimeByTrainCode()方法；如果选择的是站站查询，则使用第三方 Web 服务提供的 getStationAndTimeByStationName()方法。

12.6　天 气 查 询

好吧，出行的交通问题在上节得到了解决，可是还有一个很大的问题，那就是天气状况。本来高高兴兴地去旅游了，可是到达目的地后才发现，阴天下雨的，也没办法出去玩，只能窝在旅馆里，多扫兴呀。于是，本节就教大家写一个关于天气查询的程序，输入想要查询天气的城市，这个城市未来三天的天气情况就一目了然了，很方便吧，赶快来看看吧。

**视频教学：**光盘/videos/12/天气查询.avi　　　　长度：7 分钟

12.6.1　实例描述

五一放假了，我和家人商量着去外地旅游，因为不知道目的地的天气状况，也不知道该穿厚点还是薄点，真为难。正想着呢，一下子想起来前段时间做的一个程序：天气查询系统，哈，这样就简单多了，输入城市名，天气状况很快就了解了，而且关于这个城市的介绍和今日指数都很详细呢。

12.6.2 实例应用

【例 12-6】 天气查询

(1) 打开 12.1 节创建的 ASP.NET 项目，添加本节所需的第三方 Web 服务引用，Web 服务的 URL 地址为“http://webservice.webxml.com.cn/WebServices/WeatherWebService.asmx”，Web 引用名为“weatherServices”。

(2) 添加名为 Weather.aspx 的页面，页面前台布局效果如图 12-12 所示。



图 12-12 前台页面布局

(3) Weather.aspx 页面有两个下拉列表框，分别表示省份和城市，页面 Load 事件的代码如下：

```
weatherServices.WeatherWebService wws = new
weatherServices.WeatherWebService();//初始化 WeatherWebServices

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        sheng.DataSource = wws.getSupportDataSet().Tables[0].DefaultView;
                                        //获取支持的州和国内省份数据
        sheng.DataValueField = "ID";
        sheng.DataTextField = "Zone";
        sheng.DataBind();

        CityBind("1");              //默认省级行政单位“直辖市”
        weather("58367");           //默认查询上海天气状况
    }
}
```

在页面加载时，绑定 ID 为“sheng”的下拉列表框数据。CityBind()方法表示根据省份或直辖市得到城市数据，并绑定到 ID 为“shi”的下拉列表框。CityBind()方法的代码如下：

```
//根据选定的省份获取对应的城市
protected void CityBind(string zoneID)
{
    DataTable dt = wws.getSupportDataSet().Tables[1];
    DataView dv = new DataView(dt);
    dv.RowFilter = "ZoneID=" + zoneID;
    shi.DataSource = dv;
    shi.DataTextField = "Area";
    shi.DataValueField = "AreaCode";
    shi.DataBind();
}
```

(4)　下面是 ID 为“sheng”的下拉列表框的 onselectedindexchanged 事件的代码：

```
protected void sheng_SelectedIndexChanged(object sender, EventArgs e)
{
    CityBind(sheng.SelectedItem.Value.Trim());//根据选定的省份获取对应的城市
}
```

下拉列表框的 AutoPostBack 属性需要设为 true。

(5)　接下来是【查询】按钮的单击事件，代码如下：

```
//查询
protected void btnWeatherSearch Click(object sender, EventArgs e)
                                               //获取城市天气信息
{
    weather(shi.SelectedItem.Value.Trim());
}
private void weather(string city)
{
    string[] wa = wws.getWeatherbyCityName(city);
    Label1.Text = wa[10];
    Label2.Text = wa[6] + "   " +wa[5] +"   " +
        wa[7];
    Label3.Text = wa[3] + "   " + wa[12] + "   "
        + wa[14];
    Label4.Text = wa[18] + "   " + wa[17] + "   "
        + wa[19];
    Label5.Text = wa[11];
    Label6.Text = wa[22];
    Label7.Text = wa[4];
    Label8.Text = wa[0] + " / " + wa[1];
    Image1.ImageUrl = "~/images/weather/" + wa[8];
    Image2.ImageUrl = "~/images/weather/" + wa[9];
    Image3.ImageUrl = "~/images/weather/" + wa[15];
    Image4.ImageUrl = "~/images/weather/" + wa[16];
    Image5.ImageUrl = "~/images/weather/" + wa[20];
    Image6.ImageUrl = "~/images/weather/" + wa[21];
    CityPhoto.ImageUrl = "http://www.cma.gov.cn/tqyb/img/city/" + wa[3];
    CityPhoto.AlternateText = shi.SelectedItem.Text;
}
```

12.6.3　运行结果

运行 Weather.aspx 页面，结果如图 12-13 所示。



图 12-13　天气查询结果

12.6.4　实例分析



源码解析：

getWeatherbyCityName()方法详解：参数为城市中文名称(国外城市可用英文)或城市代码(不输入默认为上海市)，如：上海或 58367，如有城市名称重复请使用城市代码查询(可通过 getSupportCity 或 getSupportDataSet 获得)；返回数据：一个一维数组 String(22)，共有 23 个元素。

String(0) 到 String(4)：省份，城市，城市代码，城市图片名称，最后更新时间。String(5)到 String(11)：当天的气温，概况，风向和风力，天气趋势开始图片名称(以下称：图标一)，天气趋势结束图片名称(以下称：图标二)，现在的天气实况，天气和生活指数。String(12)到 String(16)：第二天的气温，概况，风向和风力，图标一，图标二。String(17)到 String(21)：第三天的气温，概况，风向和风力，图标一，图标二。String(22)：被查询的城市或地区的介绍。

12.7　常见问题解答

12.7.1　ASP.NET Web 服务和 ASP.NET 网站的区别



ASP.NET Web 服务和 ASP.NET 网站的区别。
网络课堂：http://bbs.itzcn.com/thread-15798-1-1.html

使用 Visual Studio 创建项目时，有 ASP.NET Web 服务和 ASP.NET 网站这两种类型。请问它们两个有什么区别？

【解决办法】

其实很容易区分它们。ASP.NET 网站就是基于.NET 的网站。

ASP.NET Web 服务可以创建基于.NET 的服务，就是一个应用于 Web 的应用程序，能向外界暴露一个 API 接口，然后可以通过 Web 来调用。

12.7.2　ASP.NET Web 服务应用程序问题



ASP.NET Web 服务应用程序问题。

网络课堂：http://bbs.itzcn.com/thread-15799-1-1.html

我在 ASP.NET Web 服务应用程序中有一个类和方法，代码如下所示：

```
public class UserInfo
    {
        public string email;
        public string name;
        public string state;
        public byte[] mybt;
    }
 [WebMethod]
        public void test(ref UserInfo [] u)
        {
            for (int i = 0; i < u.Length; i++)
            {
                u[i].email = i.ToString();
            }
        }
```

然后在窗体应用程序中调用，代码如下所示：

```
 ServiceReference1.UserInfo[] cu = new
WindowsFormsApplication1.ServiceReference1.UserInfo[6];
        private void button10 Click(object sender, EventArgs e)
        {
         ServiceReference1.Service1SoapClient service = new ServiceReference1.
             Service1SoapClient();
            service.test(ref cu);
            for (int i = 0; i < cu.Length; i++)
            {
                listBox1.Items.Add(cu[i].email);
            }
        }
```

结果总是出现错误：

```
System.Web.Services.Protocols.SoapException: 服务器无法处理请求。 --->
System.NullReferenceException: 未将对象引用设置到对象的实例。
 在 WebService1.Service1.test(UserInfo[]& u)位置 C:\Documents and Settings\
Administrator\My Documents\Visual Studio 2008\Projects\WebService1\
WebService1\Service1.asmx.cs:行号 213
   --- 内部异常堆栈跟踪的结尾 ---
```

参数如果不是数组而是对象的话就没有问题，这是什么原因啊？

【解决办法】

我想应该是以下原因。下面是你的代码：

```
ServiceReference1.UserInfo[] cu = new
WindowsFormsApplication1.ServiceReference1.UserInfo[6]
```

在这里 cu 本身初始化了，而 cu 里面每个元素却都是 null。所以，“public void test(ref UserInfo [] u)”里面的“u[i].email = i.ToString()”是有问题的。因为 u[i]都是 null，没有实例化。所以 email 还没有呢，怎么可以给 u[i].email 赋值啊。

12.7.3 用.NET 调用 Web 服务时报错



用.NET 调用 Web 服务时报错。

网络课堂：http://bbs.itzcn.com/thread-15800-1-1.html

在用.NET 调用 Web 服务时，报 FaultException 错误，提示“缺少 head”信息，该怎么解决呀？

【解决办法】

首先应该看看 WSDL 的 Header 是什么，一般是类似下面的内容：

```
<soap:Header
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
ity-secext-1.0.xsd">
<wsse:Security>
<wsse:UsernameToken>
<wsse:Username>username</wsse:Username>
<wsse:Password Type="wsse:PasswordText">password </wsse:UsernameToken>
</wsse:Security>
```

如果使用的是 Visual Studio 2008，可以通过 Add Service 来添加 Web 引用，然后调用时指定 username 和 password：

```
DocumentService.documentServiceClient client = new
DocumentService.documentServiceClient();
client.ChannelFactory.Credentials.UserName.UserName = "MyUserName";
client.ChannelFactory.Credentials.UserName.Password = "MyPass";
```

12.8 习　　题

一、填空题

(1) 如果要调用第三方 Web 服务提供的方法生成验证码，应该使用__________作为返回数据类型。

(2) 使用 RadioButtonList 控件时，要想实现该控件的 OnSelectedIndexChanged 事件，使选定内容更改时自动回发到服务器上，必须先设定属性__________。

二、选择题

(1) 下列代码中，横线处应填写__________。

```
ValidateCodeWS.ValidateCodeWebService vcws = new
ValidateCodeWS.ValidateCodeWebService();//实例化 Web 服务对象
protected void Page_Load(object sender, EventArgs e)
{
   if(Request.Params["validateDate"]!=null)//接收传来的参数
   {
       int i=Int32.Parse( Request.Params["validateDate"].ToString());
       Response.ContentType = "image/Png";
       byte[] b = vcws.enValidateByte(i.ToString());
       __________
   }
}
```

A. Response.BinaryWrite(b);

B. Response.WriteFile(b);

C. Response.Write(b);

D. Response.WriteSubstitution(b);

(2) 对下段代码描述正确的是__________。

```
ChinaZipSearchWebServices.ChinaZipSearchWebService ChinaZip =
 new ChinaZipSearchWebServices.ChinaZipSearchWebService();
   DataSet ds = ChinaZip.getAddressByZipCode(txtZip.Text.Trim(), "");
   GridViewShowAddress.DataSource = ds.Tables[0].DefaultView;
   GridViewShowAddress.DataBind();
```

A. 这段代码不正确

B. 在初始化时可以不用添加 ChinaZipSearchWebServices Web 引用名

C. 我们可以将代码直接写成下列形式

```
ChinaZipSearchWebServices.ChinaZipSearchWebService ChinaZip =
 new ChinaZipSearchWebServices.ChinaZipSearchWebService();
   DataSet ds = ChinaZip.getAddressByZipCode(txtZip.Text.Trim(), "");
   GridViewShowAddress.DataSource = ds.Tables[0] .Columns;
   GridViewShowAddress.DataBind();
```

D. 我们可以将代码直接写成下列形式

```
ChinaZipSearchWebServices.ChinaZipSearchWebService ChinaZip =
 new ChinaZipSearchWebServices.ChinaZipSearchWebService();
   GridViewShowAddress.DataSource =
ChinaZip.getAddressByZipCode(txtZip.Text.Trim(), "");

  GridViewShowAddress.DataBind();
```

三、上机练习

上机练习 1： 简繁体转换。

使用第三方 Web 服务，实现简单的简繁体转换小程序。简繁体转换所引用的 Web 服务的 URL 地址为“http://webservice.webxml.com.cn/WebServices/TraditionalSimplifiedWeb Service.asmx”。实现效果如图 12-14 所示。

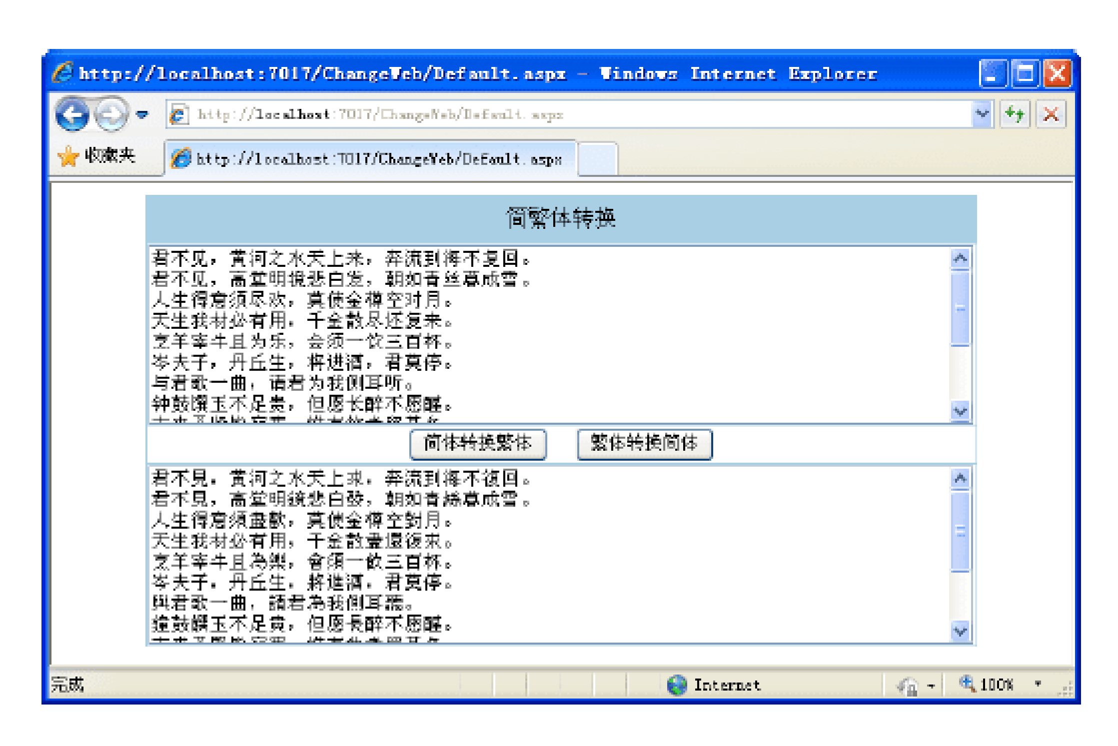
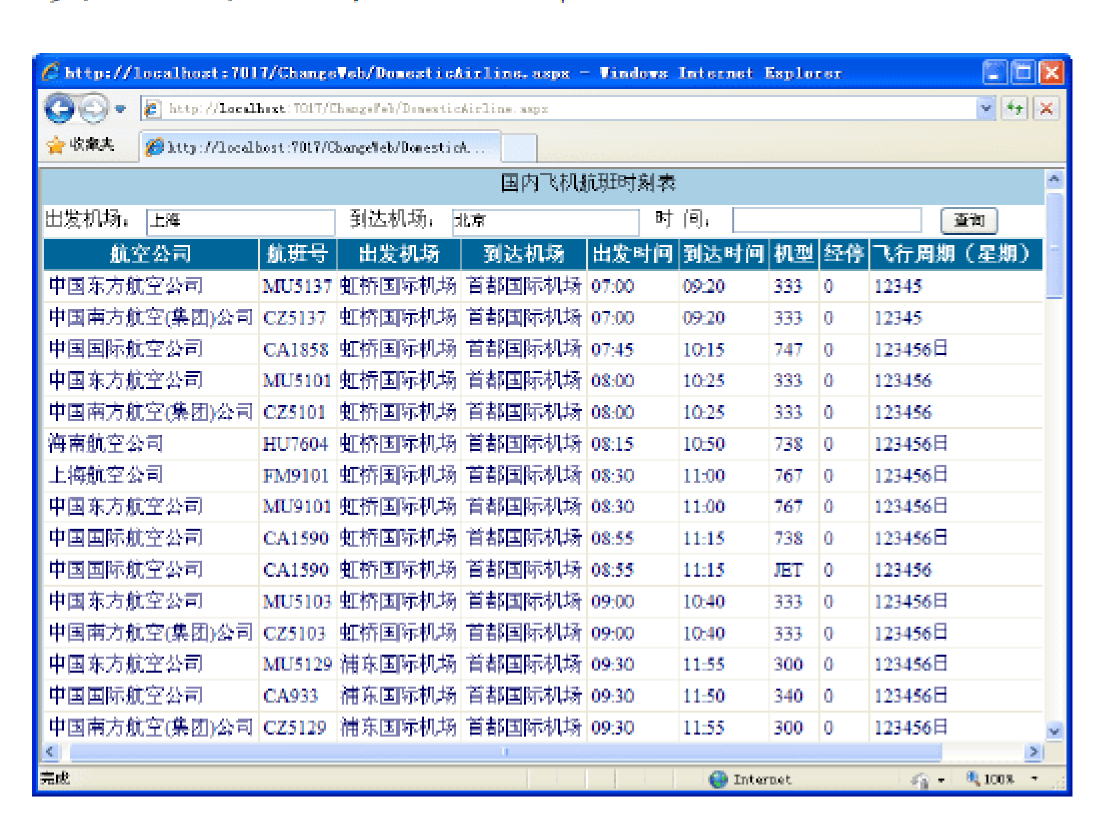


图 12-14　简体转换成繁体

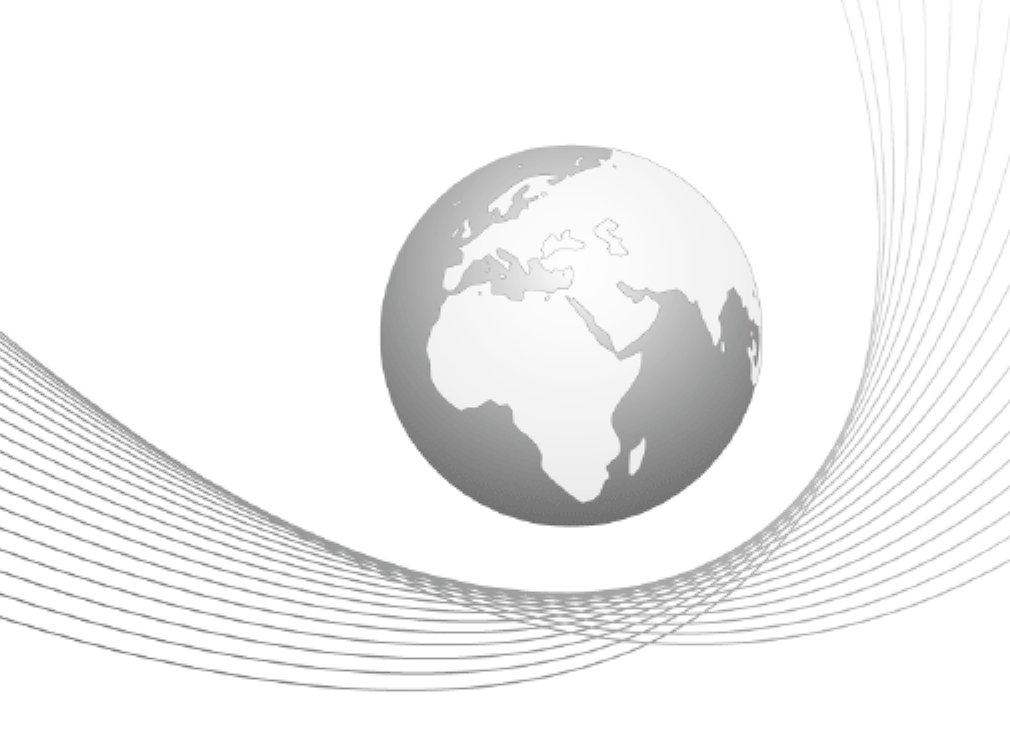
上机练习 2： 国内飞机航班时刻表。

使用第三方 Web 服务，实现国内飞机航班查询，输入出发机场和到达机场，即可查询所有航班信息。第三方 Web 服务的 URL 地址为“http://webservice.webxml.com.cn/webservices/DomesticAirline.asmx”，实现效果如图 12-15 所示。



航空公司	航班号	出发机场	到达机场	出发时间	到达时间	机型	经停	飞行周期（星期）
中国东方航空公司	MU5137	虹桥国际机场	首都国际机场	07:00	09:20	333	0	12345
中国南方航空(集团)公司	CZ5137	虹桥国际机场	首都国际机场	07:00	09:20	333	0	12345
中国国际航空公司	CA1858	虹桥国际机场	首都国际机场	07:45	10:15	747	0	123456日
中国东方航空公司	MU5101	虹桥国际机场	首都国际机场	08:00	10:25	333	0	123456
中国南方航空(集团)公司	CZ5101	虹桥国际机场	首都国际机场	08:00	10:25	333	0	123456
海南航空公司	HU7604	虹桥国际机场	首都国际机场	08:15	10:50	738	0	123456日
上海航空公司	FM9101	虹桥国际机场	首都国际机场	08:30	11:00	767	0	123456日
中国东方航空公司	MU9101	虹桥国际机场	首都国际机场	08:30	11:00	767	0	123456日
中国国际航空公司	CA1590	虹桥国际机场	首都国际机场	08:55	11:15	738	0	123456日
中国国际航空公司	CA1590	虹桥国际机场	首都国际机场	08:55	11:15	JET	0	123456
中国东方航空公司	MU5103	虹桥国际机场	首都国际机场	09:00	10:40	333	0	123456日
中国南方航空(集团)公司	CZ5103	虹桥国际机场	首都国际机场	09:00	10:40	333	0	123456日
中国东方航空公司	MU5129	浦东国际机场	首都国际机场	09:30	11:55	300	0	123456日
中国国际航空公司	CA933	浦东国际机场	首都国际机场	09:30	11:50	340	0	123456日
中国南方航空(集团)公司	CZ5129	浦东国际机场	首都国际机场	09:30	11:55	300	0	123456日

图 12-15　国内飞机航班查询结果



第 13 章　WCF 快速入门

内容摘要：

WCF 是一个基于 SOA(Service Oriented Architecture，面向服务架构)的.NET 平台下的统一框架，它代表了软件架构与开发的一种发展方向。

使用 WCF 开发人员可以用最少的时间建立软件与外界通信的模型。它整合了.NET 平台下所有和分布式系统有关的技术，像 Web Service、.NET Remoting、WSE 和 MSMQ 等。这使得开发者能够建立一个跨平台的、安全的、可依赖的、事务性的解决方案，并且能与已有系统兼容协作。

通过本章的学习，我们可以了解什么是 WCF、WCF 的基本术语以及如何创建 WCF 和编写客户端代码。

学习目标：

- 了解 WCF 的作用和组成部分
- 掌握创建 WCF 项目的方法
- 掌握如何生成 WCF 服务代理类
- 掌握客户端调用 WCF 的方法
- 理解 WCF 中地址、绑定和合约的作用和设置方法
- 掌握如何通过端点来配置 WCF 服务
- 了解如何创建 WCF 主机
- 熟悉 WCF 的开发流程

13.1 什么是 WCF

WCF 是 Windows Communication Foundation(Windows 通信基础)的简称，由.NET Framework 3.0 开始引入，与 Windows Presentation Foundation 及 Windows Workflow Foundation 共同组成了 Windows 操作系统的三个重大应用程序开发类库。

视频教学：光盘/videos/13/什么是 WCF.avi　　长度：17 分钟

13.1.1 基础知识——WCF 概述

WCF 是一个面向服务编程的分布式架构。它是.NET 框架的一部分，因为 WCF 并不能脱离.NET 框架而单独存在。因此，虽然 WCF 是微软为应对 SOA 解决方案的开发需求专门推出的，但它并不是像 Spring 和 Struts 那样的框架，也不是像 EJB 那样的容器或者服务器。

严格地说，WCF 就是专门用于服务定制、发布、运行以及消息传递和处理的一组类的集合，也就是所谓的“类库”。这些类通过一定方式被组织起来，共同协作，并为开发者提供统一的编程模式。WCF 之所以特殊，在于它所应对的场景与普通的.NET 类库不同，它主要用于处理进程甚至主机之间消息的传递与处理，同时它引入了 SOAP 的设计思想，以服务的方式公布并运行，以方便客户端跨进程和主机对服务进行调用。

实际上，WCF 就是微软对分布式编程技术的最大集成者，它将 DCOM、.NET Remoting、Enterprise Service、Web Service、WSE 以及 MSMQ 集成在一起，从而降低了学习分布式系统开发的难度，并统一了开发标准。

也就是说，在 WCF 框架下，开发基于 SOA 的分布式系统变得容易了。微软将所有与此相关的技术要素都包含在内，掌握了 WCF，就相当于掌握了敲开 SOA 大门的钥匙。

例如，下面通过一个实例说明 WCF 的优势所在。假设，我们要为一家婚车租赁公司开发一个新的应用程序，用于租车预约服务。该租车预约服务会被多种应用程序访问，包括呼叫中心，基于 J2EE 的租车预约服务以及合作伙伴的应用程序，如图 13-1 所示。

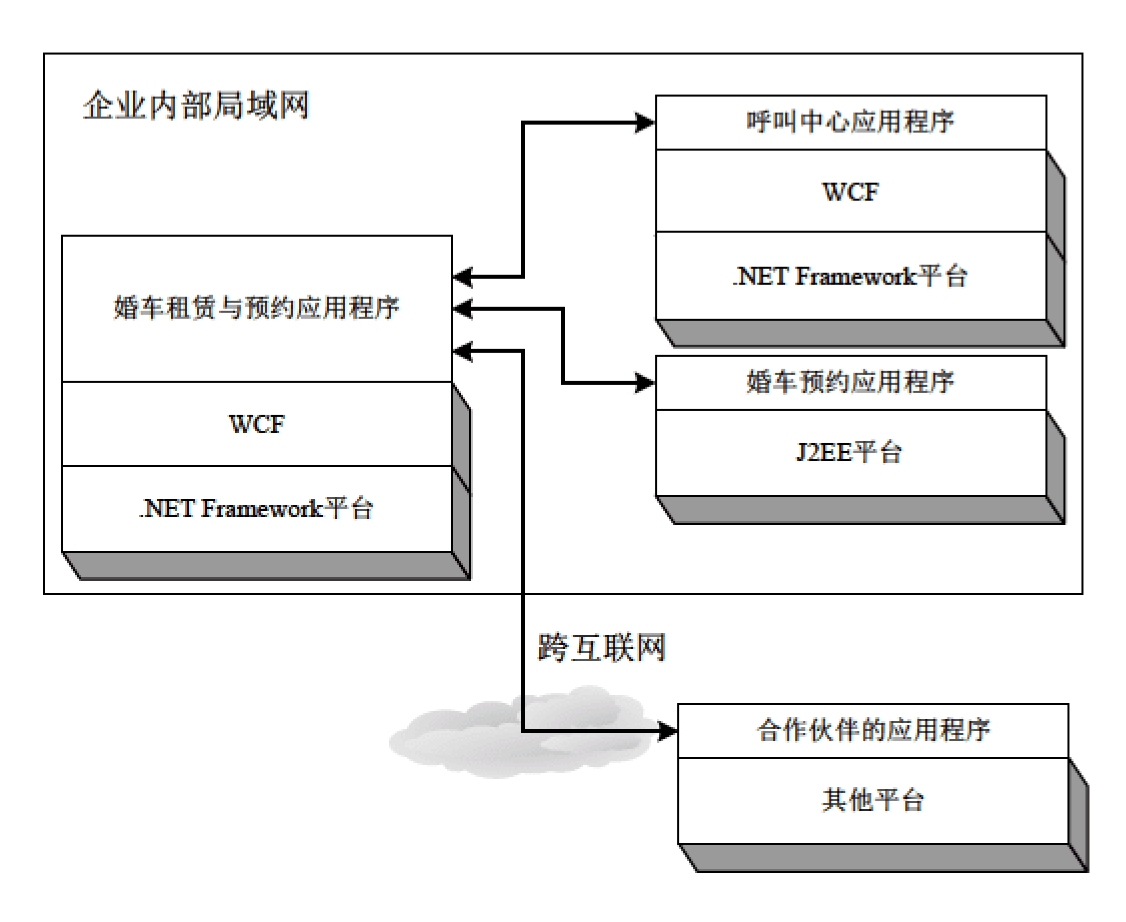


图 13-1　婚车租赁公司应用程序分布图

呼叫中心运行在 Windows 平台下，是在.NET Framework 下开发的应用程序，用户为公司员工。由于该婚车租赁公司兼并了另外一家租赁公司，该公司原有的婚车预约服务应用程序是 J2EE 应用程序，运行在非 Windows 操作系统下。呼叫中心和已有的婚车预约应用程序都运行在企业内部局域网环境下。合作伙伴的应用程序可能会运行在各种平台下，这些合作伙伴包括婚纱拍摄公司、司仪策划公司等，他们会通过 Internet 来访问婚车预约服务，实现对婚车的租用。

这样一个案例是一个典型的分布式应用系统。如果没有 WCF，利用.NET 现有的技术应该如何开发呢？

首先考虑呼叫中心，它和我们要开发的婚车预约服务一样，都是基于.NET Framework 的应用程序。呼叫中心对于系统的性能要求较高，在这样的前提下，.NET Remoting 是最佳的实现技术。它能够高性能地实现.NET 与.NET 之间的通信。

要实现与已有的 J2EE 婚车预约应用程序之间的通信，只有基于 SOAP 的 Web Service 才可以，它保证了跨平台的通信；而合作伙伴由于是通过 Internet 来访问，利用 ASP.NET Web Service，也是较为合理的选择，它保证了跨网络的通信。由于涉及到网络之间的通信，我们还要充分考虑通信的安全性，利用 WSE(Web Service Enhancements)可以为 Web Service 提供安全的保证。

一个好的系统除了要保证访问和管理的安全和高性能外，同时还要保证系统的可依赖性。因此，事务处理是企业应用必须考虑的因素，对于婚车预约服务同样如此。在.NET 中，Enterprise Service(COM+)提供了对事务的支持，其中还包括分布式事务(Distributed Transactions)。不过对于 Enterprise Service 而言，它仅支持有限的几种通信协议。

如果还要考虑异步调用、脱机连接、断点连接等功能，我们还需要应用 MSMQ(Microsoft Message Queuing)利用消息队列支持应用程序之间的消息传递。

如此看来，要建立一个好的婚车租赁预约服务系统，需要用到的.NET 分布式技术包括：.NET Remoting、Web Service、COM+等 5 种技术，这既不利于开发者的开发，也加大了程序的维护难度和开发成本。正是基于这样的缺陷，WCF 才会在.NET 3.0 中作为全新的分布式开发技术被微软强势推出，它整合了上述介绍的分布式技术，成为理想的分布式开发解决方案。如表 13-1 所示列出了 WCF 与之前相关技术的比较。

表 13-1　WCF 与现有技术比较

特　性	Web Service	.NET Remoting	Enterprise Services	WSE	MSMQ	WCF
具有互操作性的 Web 服务	支持					支持
支持.NET 之间的通信		支持				支持
分布式事务			支持			支持
支持 WS 标准				支持		支持
消息队列					支持	支持

从表 13-1 中来看，WCF 完全可以看做是 Web Service、.NET Remoting、Enterprise Service、WSE、MSMQ 等技术的并集。因此，对于上述婚车预约服务系统的例子，利用 WCF，就可以解决包括安全、可依赖、互操作、跨平台通信等需求。开发者再不用去分别了解.NET Remoting

和 WSE 等各种技术了。



事实上 WCF 远非简单的并集这样简单，它是真正面向服务的产品，它已经改变了通常的开发模式。

13.1.2 基础知识——WCF 组成部分

软件设计的一个重要原则：软件组件必须针对特定的任务专门设计和优化。假如我们要做一个管理软件，想象一下，如果一个软件非常依赖于与外界通信，那么我们不能把管理软件与外界通信的逻辑考虑在管理系统内部。所以，必须把通信任务委托给不同的组件。用 WCF 术语来说，这个组件称为 WCF 服务。更通俗地讲，WFC 服务就是负责与外界通信的软件。

一个 WCF 服务由下面三部分组成。

- Service Class：一个标记了 ServiceContract 属性的类，其中可以包含多个方法。该类除了标记一些 WCF 特有的属性外，与一般的类没有区别。
- Host：可以是应用程序、进程或者 Windows 服务等，它是 WCF 服务的运行环境。
- Endpoints：可以是一个，也可以是一组，它是 WCF 实现通信的核心要素。

如图 13-2 显示了这三部分在 WCF 服务中的位置。

一个 WCF 服务必须能为不同的通信场景提供不同的访问点，这些访问点称为 WCF 端点，也就是上面所提到的 EndPoint。每个端点都有一个绑定(Binding)、一个地址(Address)和一个合约(Contract)，如图 13-3 所示。

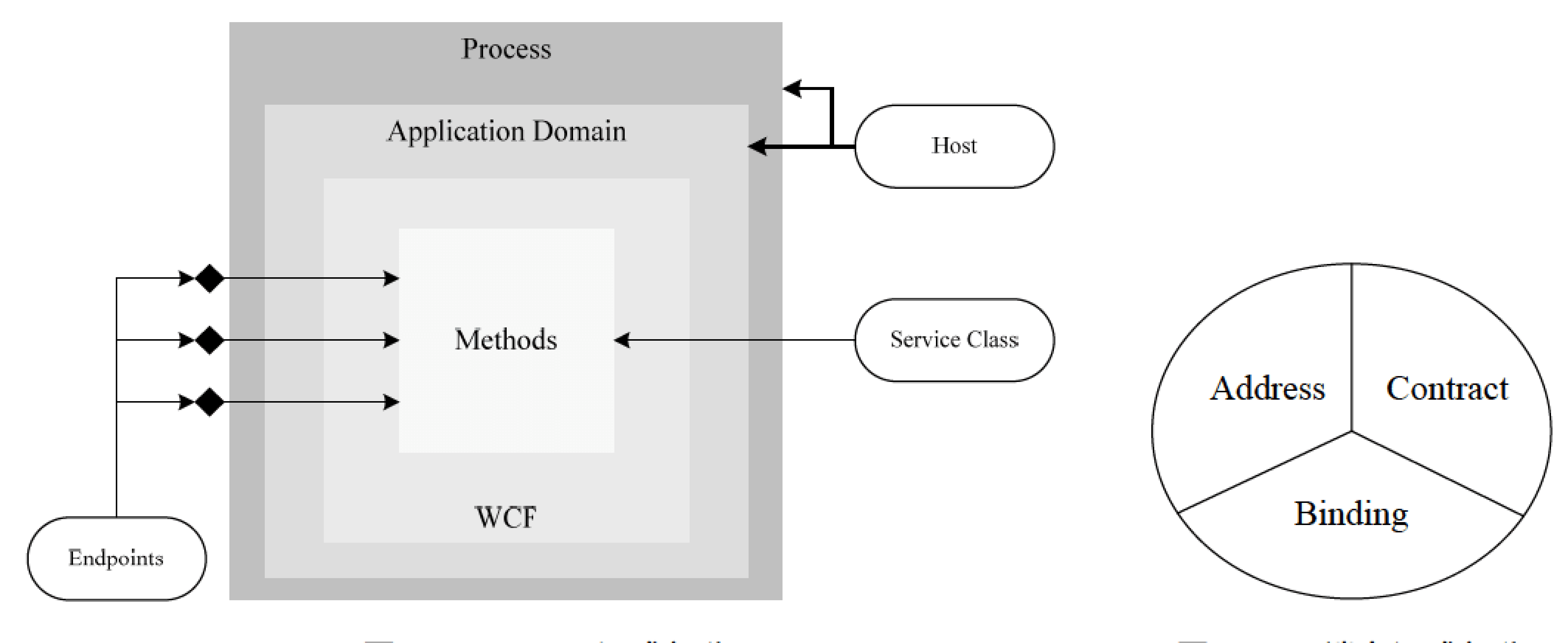


图 13-2　WCF 组成部分　　图 13-3　端点组成部分

- 绑定：指定该端点如何与外界通信，也就是为端点指定通信协议，包括传输协议、编码协议和安全协议。
- 地址：一个端点地址，如果通过端点与 WCF 通信，必须把通信指定到网络地址。
- 合约：一个端点上合约指定通过该端点的用户能访问到 WCF 服务的什么操作。



为了便于记忆 EndPoint 的这三个部分，可以将 Address、Binding 和 Contract 简称为“ABC”。

13.2　创建第一个 WCF 服务程序

通过上节的学习，相信大家对 WCF 的概念、作用及其组成部分都有了一个大致的了解。本节打算在详细介绍 WCF 之前看一个简单的 WCF 实例，通过本实例我们将了解 WCF 程序的开发步骤和运行流程，为后面核心概念的理解奠定基础。

**视频教学：**光盘/videos/13/第一个 WCF 程序.avi　　长度：16 分钟

13.2.1　实例描述

由于 WCF 是在.NET Framework 3.0 之后开始出现的，所以在开发之前首先要选择一款支持它的开发工具。这里我们以 Visual Studio 2010 为例进行介绍。

在 Visual Studio 2010 中创建 WCF 服务的过程非常简单，我们不需要做任何配置，只需要根据需求编写相应的业务逻辑代码即可。

13.2.2　实例应用

【例 13-1】　创建第一个 WCF 服务程序

(1)　在 Visual Studio 2010 中打开【新建项目】对话框，选择模板为 WCF，再选择【WCF 服务库】项，然后设置项目名称为“FirstWcfServiceLibrary”，单击【确定】按钮完成创建，如图 13-4 所示。

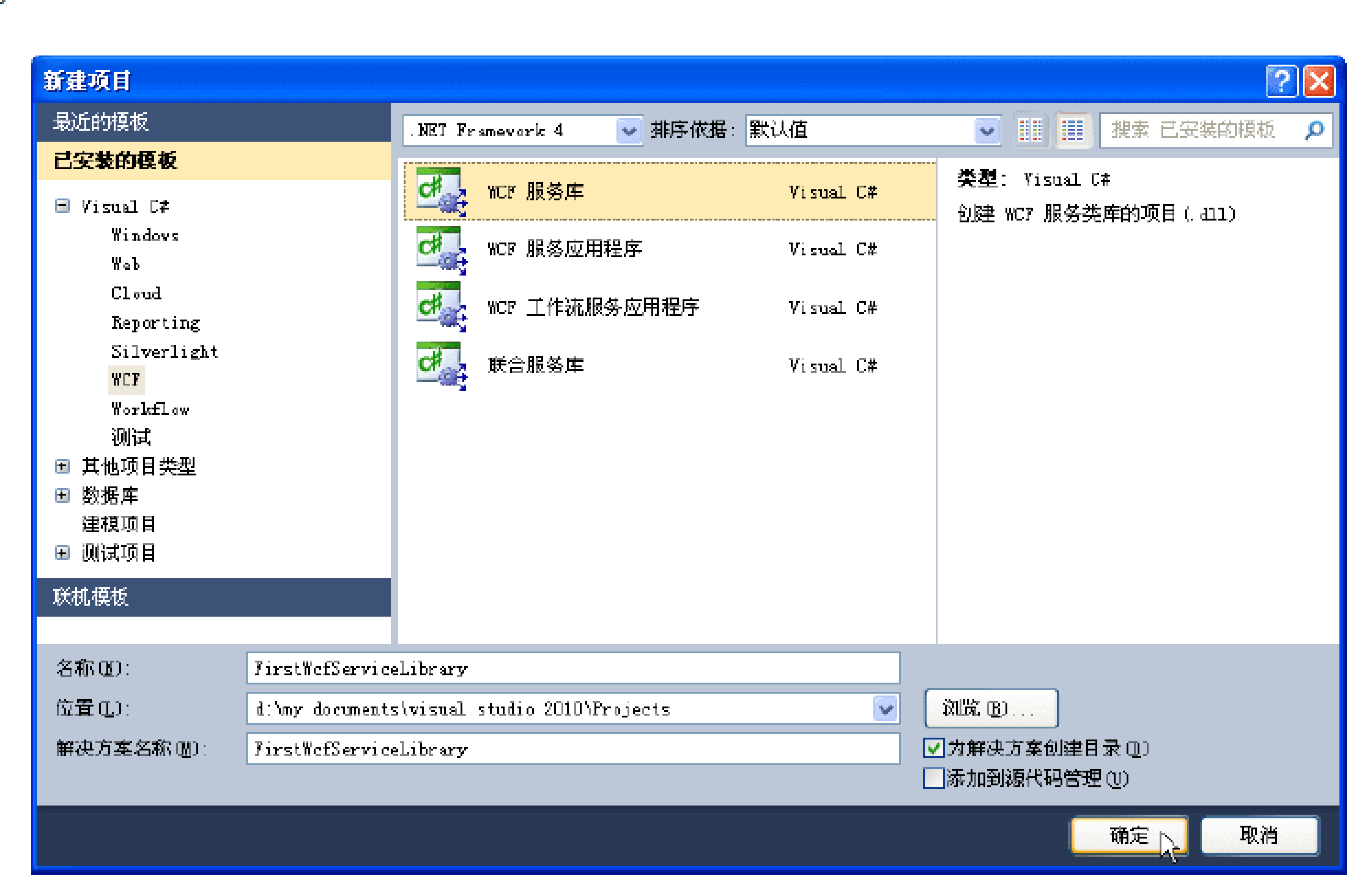


图 13-4　创建 WCF 项目

(2)　WCF 项目创建完成之后，在【解决方案资源管理器】窗口中将看到默认生成的项目结构及 IService1.cs 文件的内容，如图 13-5 所示。

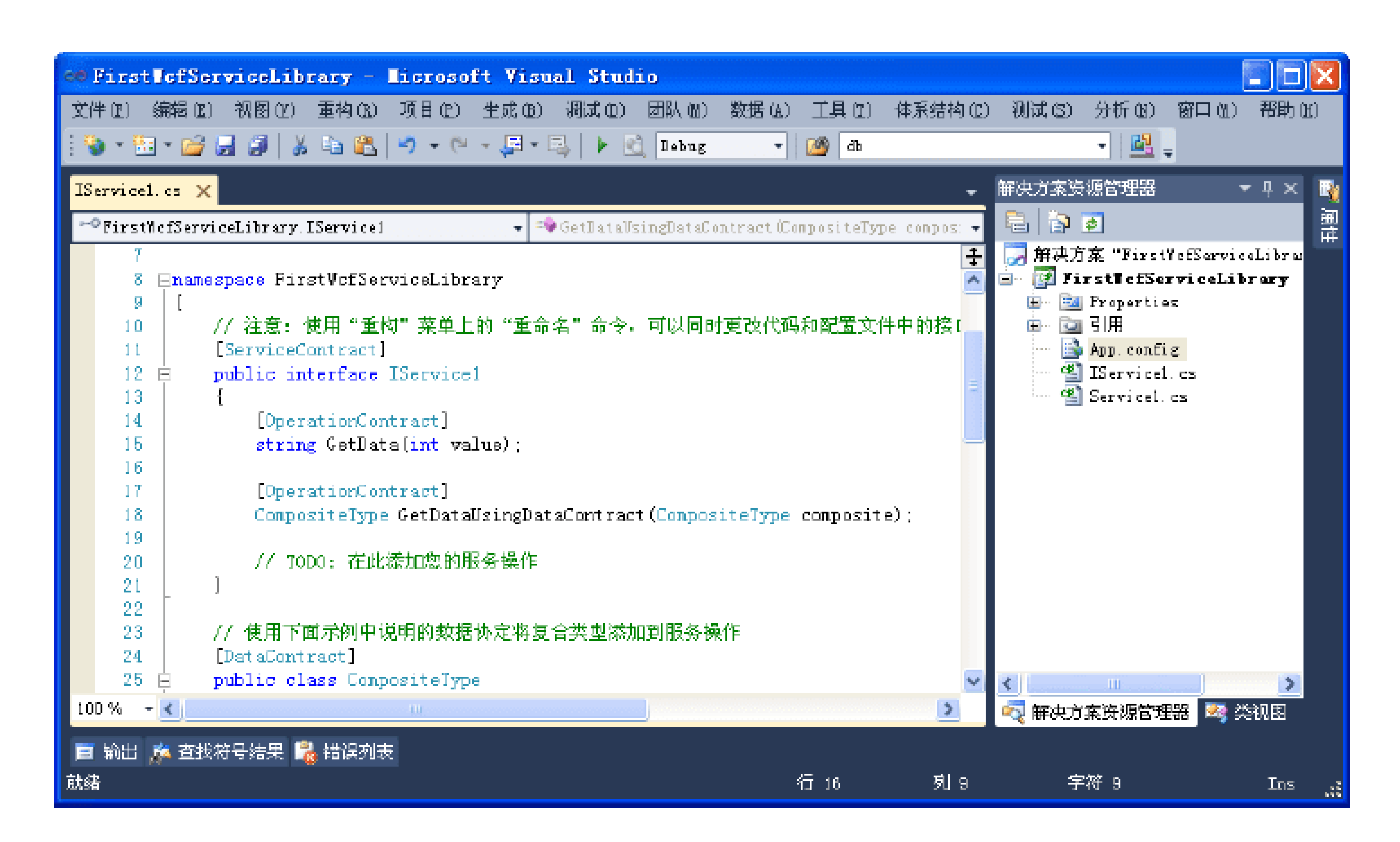


图 13-5　WCF 项目内容

在这里的 IService1.cs 称为合约，WCF 的合约必须要以接口的方式定义。Service1.cs 为服务的实现，即它继承接口并进行实现。App.config 是 WCF 服务的配置信息。

(3)　在 IService1.cs 文件中的“[ServiceContract]”为服务合约，“[OperationContract]”声明该方法可以在 WCF 中调用。按照这种规则，手动创建一个方法，代码如下：

```
//手动创建的方法
[OperationContract]
string Welcome(string str);
```

(4)　打开 Service1.cs 文件，添加服务合约中声明的 Welcome()方法的实现代码。具体代码如下所示：

```
//实现在服务合约中定义声明的 Welcome()方法
public string Welcome(string str)
{
    return string.Format("你好 {0}，欢迎来到 WCF 的世界。当前时间是：{1}。", str,
    DateTime.Now.ToString());
}
```

(5)　至此，关于 WCF 服务合约的创建及实现就都定义好了。下面对 WCF 服务的配置信息进行定义，指定客户端通过什么方式引用 WCF 服务。打开 App.config 文件，其中的内容如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <!-- 部署服务库项目时，必须将配置文件的内容添加到主机的 app.config 文件中。
    System.Configuration 不支持库的配置文件。-->
  <system.serviceModel>
    <services>
      <service name="FirstWcfServiceLibrary.Service1">
```

```
        <host>
          <baseAddresses>
            <add baseAddress = "http://localhost:8732/Design_Time_Addresses/
                FirstWcfServiceLibrary/Service1/" />
          </baseAddresses>
        </host>
        <!-- Service Endpoints -->
        <!-- 除非完全限定，否则地址将与上面提供的基址相关 -->
        <endpoint address ="" binding="wsHttpBinding"
             contract="FirstWcfServiceLibrary.IService1">
          <!--
              部署时，应删除或替换下列标识元素，以反映用来运行所部署服务的标识。
              删除之后，WCF 将自动推断相应标识。
          -->
          <identity>
            <dns value="localhost"/>
          </identity>
        </endpoint>
        <!-- Metadata Endpoints -->
        <!-- 元数据交换终节点供相应的服务用于向客户端做自我介绍。 -->
        <!-- 此终节点不使用安全绑定，应在部署前确保其安全或将其删除-->
        <endpoint address="mex" binding="mexHttpBinding"
             contract="IMetadataExchange"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- 为避免泄漏元数据信息，
          请在部署前将以下值设置为 false 并删除上面的元数据终节点  -->
          <serviceMetadata httpGetEnabled="True"/>
          <!-- 要接收故障异常详细信息以进行调试，请将以下值设置为 true。
          在部署前设置为 false 以避免泄漏异常信息-->
          <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

(6) 在这个文件中需要修改的是 baseAddresses 节点下 add 子节点的 baseAddress 属性，该属性为一个 URL 用于指定 WCF 的引用地址。如下所示为本实例中修改后的代码：

```
    <baseAddresses>
      <add baseAddress =
"http://localhost:8732/FirstWcfServiceLibrary/Service1/" />
    </baseAddresses>
```

(7) 下面创建一个控制台应用程序，通过代码的形式启动一个 WCF 服务主机。该控制台应用程序与 WCF 项目在同一个解决方案中，并引用了 WCF 项目，名称为“testWCFConsoleApplication”。

(8) 在控制台中添加如下命名空间的引用：

```
using System.ServiceModel;
using FirstWcfServiceLibrary;
```



System.ServiceModel 命名空间是 WCF 服务必需的，如果不引用将无法调用 WCF。

(9) 对控制台中的 Main()方法进行修改，最终代码如下所示：

```
static void Main(string[] args)
{
    //指定主机使用 WCF 服务的 URL
    Uri wcfUri=new Uri("http://localhost:8732/FirstWcfService");
    //针对 WCF 服务创建一个主机
    ServiceHost host = new ServiceHost(typeof(Service1), wcfUri);
    //打开主机
    host.Open();
    //输出提示信息
    Console.WriteLine("WCF 服务已经在主机启动。");
    Console.WriteLine("按任意键结束服务!");
    Console.Read();
    //关闭主机
    host.Close();
}
```

(10) 下面创建一个控制台应用程序作为 WCF 客户端，它也与 WCF 项目在同一个解决方案中，名称为“WCFServiceClient”。

(11) 大家还记得调用 Web 服务时的代理类吧(使用 wsdl 命令生成的.cs 文件)。同样，在调用 WCF 服务之前我们也要生成一个代理类。

从 Visual Studio 2010 的开始菜单项中选择【命令提示】工具进入命令行。然后转到客户端要存放代理类的目录，在本实例中为“D:\WCF”。

(12) 运行 svcutil 命令生成代理类。此时需要指定生成时的语言、类名、配置文件以及 WCF 服务的 URL 地址，执行成功之后将生成两个文件，如图 13-6 所示。

图 13-6　使用 svcutil 生成 WCF 代理类

(13) 在 WCFServiceClient 项目中使用添加现有项功能，将上步生成的两个文件添加到项目中。

(14) 添加对“System.ServiceModel”和“System.Runtime.Serialization”程序集的引用。

(15) 将代理类生成的命名空间“FirstWcfServiceLibrary”添加到项目中。

```
using FirstWcfServiceLibrary;
```

(16) 在 Main()方法中编写客户端代码，最终代码如下所示：

```
 static void Main(string[] args)
 {
     //实例化 WCF 中的类
     Service1Client client = new Service1Client();
     //输出提示信息
     Console.WriteLine("-------------------欢迎使用 WCF 客户端应用程序
-------------------");
     Console.WriteLine();
     Console.Write("请输入名字：");
     string username = Console.ReadLine();
     //调用 Welecome()方法
     string result = client.Welcome(username);
     Console.WriteLine();
     Console.WriteLine("WCF 服务器端返回的结果为：");
     //输出结果
     Console.WriteLine(result);
     Console.WriteLine();
     Console.WriteLine("按任意键退出程序!");
     Console.ReadKey();
 }
```

(17) 至此，完成了第一个 WCF 服务的编写、测试与调用。整个解决方案中包含了一个 WCF 项目和两个控制台项目。

13.2.3　运行结果

现在，我们开始运行上面创建的程序。首先是运行 WCF 项目，在 FirstWcfServiceLibrary 中按 F5 键执行。

此时，Visual Studio 2010 检测到当前要运行的 WCF 项目，它会打开【WCF 测试客户端】工具。在这里可以浏览 WCF 服务的地址、提供的方法、配置文件，而且还可以对方法进行测试。如图 13-7 所示为测试 Welcome()方法时的窗口。

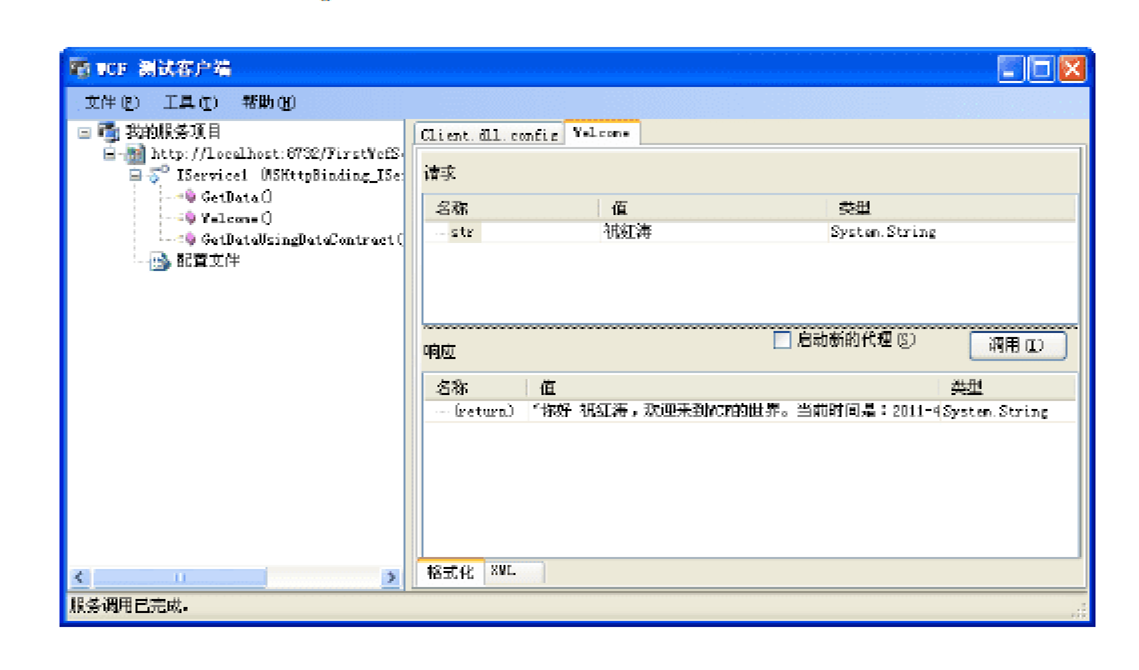


图 13-7　使用【WCF 测试客户端】测试 Welcome()方法

与 Web 服务类似，WCF 服务也可以在浏览器中浏览。例如，在本实例中可以通过地址“http://localhost:8732/FirstWcfServiceLibrary/Service1/”来浏览 WCF 服务，如图 13-8 所示。



图 13-8 使用浏览器查看 WCF 服务

如果输入地址“http://localhost:8732/FirstWcfServiceLibrary/Service1/?WSDL”将可以看到 WCF 服务的 WSDL 内容，如图 13-9 所示。

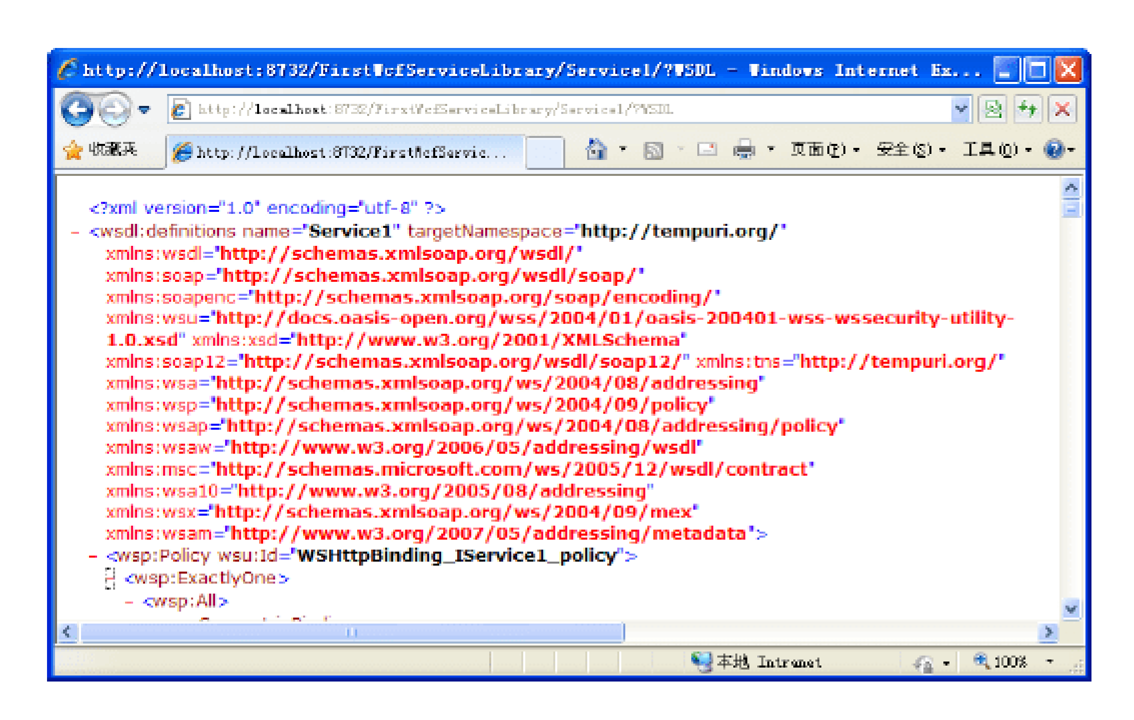


图 13-9 查看 WCF 服务的 WSDL

现在切换到 testWCFConsoleApplication 项目并运行。该项目的作用是创建一个主机并在指定的 URL 中开启 WCF 服务，运行后的输出如图 13-10 所示。

在程序中我们指定从 URL“http://localhost:8732/FirstWcfService”启动 WCF 服务，运行后可以在浏览器中输入该 URL 验证是否成功，如图 13-11 所示。

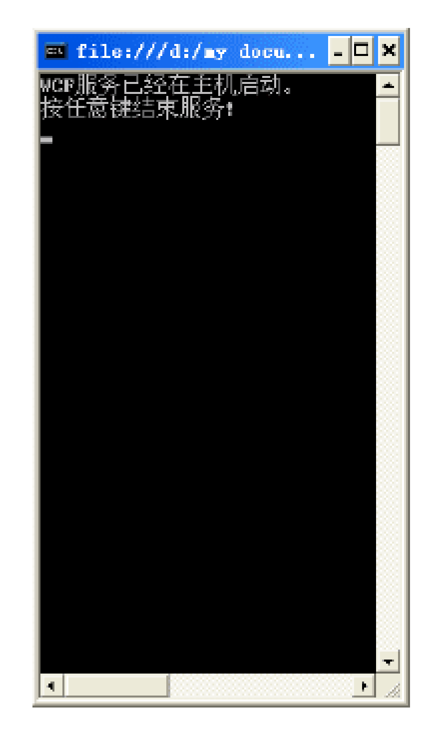


图 13-10 启动 WCF 服务

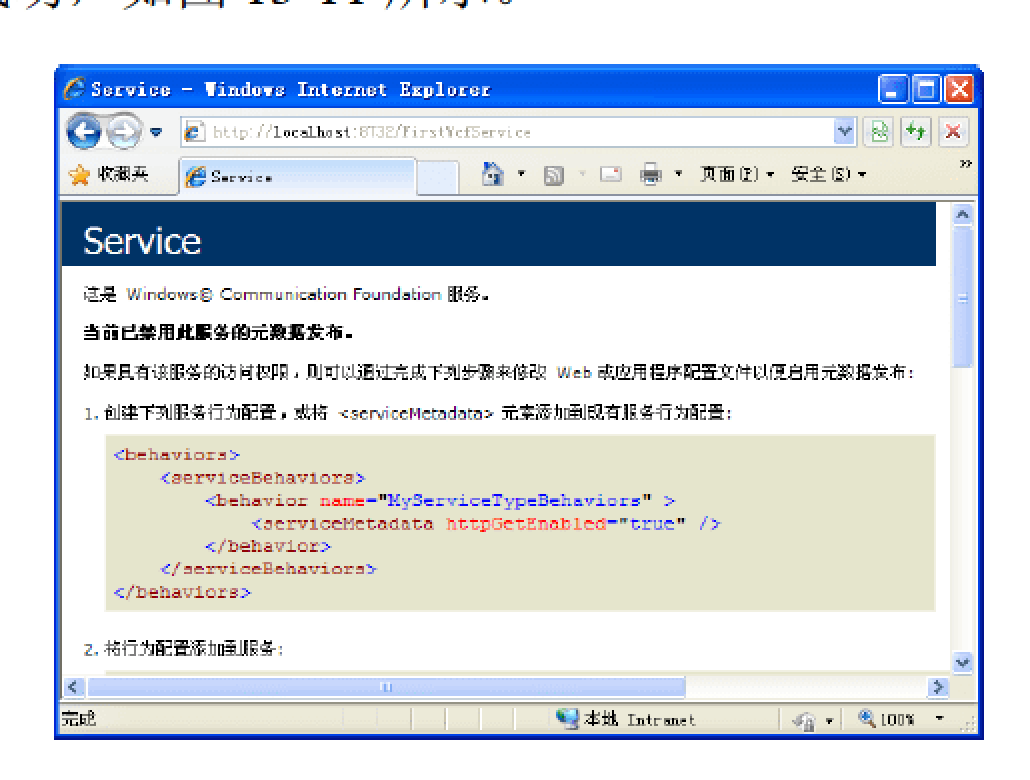


图 13-11 验证 WCF 服务

转到 WCFServiceClient 项目中，启动它来验证生成的代理类是否有效。如果有效将会看到类似图 13-12 所示的输出结果。

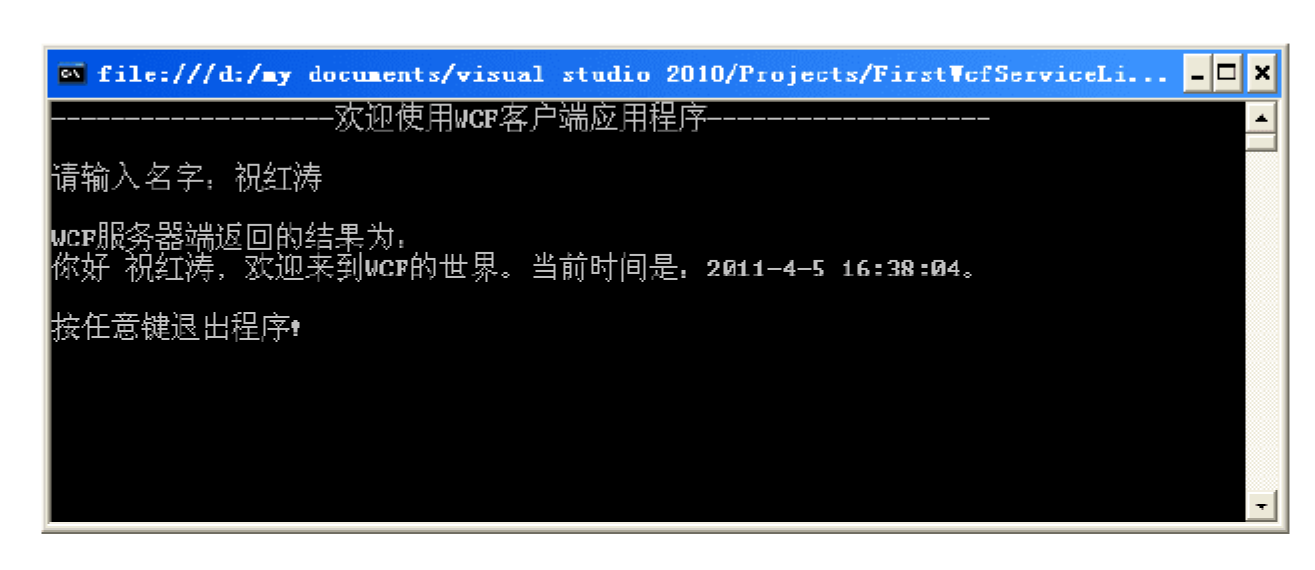


图 13-12　使用代理类测试 WCF 服务

13.2.4　实例分析



源码解析：

本实例主要讲解了 WCF 服务的开发和使用过程，而未对涉及的术语展开介绍。

为了演示这个过程，在本实例中创建了 3 个项目并分别运行进行测试。第 1 个为 WCF 服务项目，Visual Studio 2010 为它提供了单独的测试工具。当然也可以通过浏览器来验证。第 2 个项目基于 WCF 服务项目来创建一个主机，它可以向外界提供 WCF 服务。第 3 个项目则是一个客户端，它通过代理类与 WCF 服务进行通信。对于第 3 个项目也可以通过添加服务引用的方式来使用，有兴趣的读者可以试试。

最后要注意，应该先定义一个接口，再实现接口，然后配置位置信息，接下来才是调用。

13.3　WCF 核心概念详解

如果仅仅是希望能够在项目中应用 WCF，那么对于读者而言，只要掌握了 WCF 的基础知识，那么对于一般的应用就足够了。要做到这一点就很容易了，微软秉承了一贯的方式，将 WCF 这门技术优雅地呈现给开发者，封装了复杂的实现逻辑，提供了易于调用的类库和相关的工具，使得开发者能够快速地完成 WCF 程序的开发。

但是，如果我们要应用 WCF 实现一整套解决方案，就需要深度挖掘 WCF 的内部实现，掌握许多 WCF 的高级应用，并了解如何合理、有效地应用 WCF，并根据项目实际情况对 WCF 进行扩展。

在这一节，我们将对前面出现的 WCF 核心概念进行详细介绍，像 WCF 地址、绑定和合约等。

**视频教学：**光盘/videos/13/ WCF 概念详解.avi　　长度：27 分钟

13.3.1 基础知识——地址

WCF 的每一个服务都具有一个唯一的地址(Address)。每个地址都包含两个重要元素：服务位置与传输协议，或者是用于服务通信的传输方式。服务位置包括目标主机名、站点(或者网络)、通信端口、管道(或者队列)，以及一个可选的特定路径或者 URI。

总的来说，WCF 支持如下的几种传输方式：

- HTTP
- TCP
- Peer Network(对等网)
- IPC(基于命名管道的内部进程通信)
- MSMQ

地址通常采用如下格式：

```
[传输协议]://[主机名或域名][:可选端口]
```

例如，如下所示的这些都是正确的地址：

```
http://localhost:8080
http://localhost:5678/MyWcfService
net.tcp://localhost:1010/MyWcfService
net.pipe://localhost/MyWcfPipe
net.msmq://localhost/public/MyWcfService
net.msmq://localhost/MyWcfService
```

如上述的示例所示，可以将地址"http://localhost:8080"理解为"采用 HTTP 协议访问 localhost 主机，并在 8080 端口等待用户的调用"。对于"http://localhost:5678/MyWcfService"地址则可以理解为"采用 HTTP 协议访问 localhost 主机，MyWcfService 服务在 5678 端口处等待用户的调用。"

1. TCP 地址

TCP 地址采用 net.tcp 作为协议进行传输，通常它还带有端口号。例如：

```
net.tcp://localhost:8018/MyWcfService
```

如果没有指定端口号，则 TCP 地址的默认端口号为 808。例如：

```
net.tcp://localhost/MyWcfService
```

另外，两个 TCP 地址(来自于相同的主机)可以共享一个端口。例如：

```
net.tcp://localhost:8018/MyWcfService
net.tcp://localhost:8018/MyOtherWcfService
```

2. HTTP 地址

HTTP 地址是使用频率最高的一种地址，它使用 HTTP 协议进行传输，也可以利用 HTTPS 进行安全传输。HTTP 地址通常会被用作对外的基于 Internet 的服务，并为其指定端口号。

例如：

```
http://localhost:8088
http://localhost:8088/MyWcfService
https://localhost/MyWcfService
```

如果没有指定端口号，则默认为 80。与 TCP 地址相似，两个相同主机的 HTTP 地址可以共享一个端口，甚至相同的计算机。

3．IPC 地址

IPC 地址使用 net.pipe 协议进行传输，这意味着它将使用 Windows 的命名管道机制。在 WCF 中，使用命名管道的服务只能接收来自同一台计算机的调用。

因此，在使用时必须明确指定本地计算机名或者直接命名为 localhost，然后再为管道名提供一个唯一的标识字符串。例如：

```
net.pipe://localhost/MyWcfPipe
net.pipe://zhht/MyWcfPipe
```



每台计算机只能打开一个命名管道。因此，两个命名管道地址在同一台计算机上不能共享一个管道名。

4．MSMQ 地址

MSMQ 地址使用 net.msmq 协议进行传输，即使用了微软消息队列(Microsoft Message Queue)机制。在使用时必须为 MSMQ 地址指定队列名。如果是处理私有队列，则必须指定队列类型。例如：

```
net.msmq://localhost/private/MyWcfService
net.msmq://localhost/private/MyOtherWcfService
```

但对于公有队列而言，队列类型可以省略。例如：

```
net.msmq://localhost/MyWcfService
net.msmq://zhht/private/MyWcfService
```

5．对等网地址

对等网地址(Peer Network Address)使用 net.p2p 协议进行传输，它使用了 Windows 的对等网传输机制。如果没有使用解析器，我们就必须为对等网地址指定对等网名称、唯一的路径以及端口。



由于对等网的使用与配置超出了本书范围。因此在这里就不再介绍，有兴趣的读者可以参考相关书籍。

13.3.2　基础知识——绑定

WCF 中的绑定(Binding)指定了服务的通信方式。

使用绑定也是 WCF 开发区别于 Web 服务开发的一个重要方面。因为 WCF 带有许多可供选择的绑定，每种绑定都适合于特定的需求。另外，如果现有的绑定类型不能满足需求，还可以通过扩展 CustomBinding 类型创建绑定。

简单来说，WCF 绑定可以指定如下特性：

- 传输协议
- 安全要求
- 编码格式
- 事务处理要求

一个绑定类型包含多个绑定元素，它们描述了上面所有的绑定要求。WCF 内置了 10 种类型的绑定，表 13-2 中列出这些及其说明。

表 13-2　WCF 绑定类型

绑定类型	说　明
BasicHttpBinding	BasicHttpBinding 在 Web 服务的交互操作中使用最广泛，可使用 HTTP 协议或者 HTTPS 协议进行传输，其安全性取决于协议安全
WSHttpBinding	WSHttpBinding 是对 BasicHttpBinding 的增强，它使用扩展的 SOAP 确保安全性、可靠性和事务处理。同样使用 HTTP 协议或者 HTTPS 协议进行传输，但是为了确保安全，使用了 WS-Security
WSFederationHttpBinding	WSFederationHttpBinding 是一种安全、可交互操作的绑定，它支持在多个系统上共享身份，以进行身份验证和授权
WSDualHttpBinding	与 WSHttpBinding 相反，这种类型支持消息的双向传输
NetTcpBinding	使用 TCP/IP 协议为通信提供绑定
NetPeerTcpBinding	使用对等网协议为通信提供绑定
NetNamedPipeBinding	使用命名管道协议为通信提供绑定，该类型为在同一系统的不同进程之间的通信进行了优化
NetMsmqBinding	该类型的绑定会将消息发送到消息队列中，它要求 WCF 应用程序位于客户端和服务器上
MsmqIntegrationBinding	该类型的绑定将会使用消息队列的已有应用程序
CustomBinding	这种类型是自定义的绑定类型



所有以 Net 前缀开始的绑定类型都使用二进制编码在.NET 应用程序之间通信，这种编码格式比文本格式要快。

在表 13-2 中列出的每种绑定类型都有自己的特性。例如，以 WS 为前缀的绑定类型是独立于平台的，支持 Web 服务规范。以 Net 为前缀的是使用二进制格式，使.NET 应用程序之间的通信具有更高的性能。如表 13-3 中针对这些绑定类型按特性进行了分类。

表 13-3 绑定类型支持的特性

特　性	支持的绑定类型
会话	WSHttpBinding、WSDualHttpBinding、WSFederationHttpBinding、NetTcpBinding、NetNamedPipeBinding
可靠的会话	WSHttpBinding、WSDualHttpBinding、WSFederationHttpBinding、NetTcpBinding
事务处理	WSHttpBinding、WSDualHttpBinding、WSFederationHttpBinding、NetTcpBinding、NetNamedPipeBinding、NetMsmqBinding、MsmqIntegrationBinding
双向通信	WSDualHttpBinding、NetTcpBinding、NetNamedPipeBinding、NetPeerTcpBinding

除了定义绑定之外，WCF 服务还必须定义端点。端点依赖于合约、服务的地址和绑定。例如，在下面的示例代码中，实例了一个 ServiceHost 对象，将地址“http://localhost:8080/MyWcfService”和一个 WSHttpBinding 实例绑定到服务的一个端点上。

```
static void StartService()
{
    ServiceHost host;
    Uri baseAddress = new Uri("http://localhost:8080/MyWcfService");
    host = new ServiceHost(typeof(Service1));
    WSHttpBinding binding = new WSHttpBinding();
    host.AddServiceEndpoint(typeof(Service1), binding, baseAddress);
    host.Open();
}
```

除了以编程方式定义绑定之外，还可以在应用程序的配置文件中定义它。WCF 的所有配置都位于<system.serviceModel>节点中，<service>节点定义了 WCF 中所提供的服务，<bindings>节点定义了绑定信息。

例如，下面的配置文件同样实现了上述代码的功能：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="FirstWcfServiceLibrary.Service1">
        <host>
          <baseAddresses>
            <add baseAddress = "http://localhost:8080/MyWcfService" />
          </baseAddresses>
        </host>
        <endpoint address ="" binding="wsHttpBinding" contract="
            FirstWcfServiceLibrary.IService1"bindingConfiguration="config1"/>
      </service>
    </services>
    <bindings>
      <wsHttpBinding>
        <binding name="config1">
          <reliableSession enabled="true"/>
```

```
        </binding>
      </wsHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

可以看到，一个 WCF 服务必须要有一个端点，该端点包含地址、绑定和合约信息。WSHttpBinding 的默认配置由 bindingConfiguration 属性指定，该属性引用了下方名为“config1”的绑定配置信息。该配置信息位于<bindings>节点中，并启用了 reliableSession。

13.3.3 基础知识——合约

任何一个分布式应用程序，它之所以能够互相传递消息，都是事先制定好了数据交换规则，这个规则正是交换数据的双方(比如服务器端和客户端)能彼此理解对方的依据。WCF 作为分布式开发技术的一种，同样具有这样一种特性。而在 WCF 中制定的规则就被称之为合约(Contract)，它是 WCF 的消息标准，是任何一个 WCF 程序都不可或缺的一部分。

在 WCF 中合约分为 4 种，分别为：定义服务操作的服务合约(Service Contract)、定义自定义数据结构的数据合约(Data Contract)、定义错误异常的异常合约(Fault Contract)，以及直接控制消息格式的消息合约(Message Contract)。

1. 服务合约

一般情况下，我们用接口(Interface)来定义服务合约。虽然我们也可以使用类(Class)来定义，但使用接口的好处更明显一些。主要表现在如下方面。

- 便于合约的继承，不同的类型可以自由实现相同的合约。
- 同一服务类型可以实现多个合约。
- 和接口隔离原则相同，我们随时可以修改服务类型。
- 便于制定版本升级策略，让新老版本的服务合约同时使用。

服务合约定义了 WCF 服务可以执行的操作，它又包括 ServiceContract 和 OperationContract 两种。ServiceContract 用于类或者接口上，用于指定此类或者接口能够被远程调用，而 OperationContract 用于类中的方法上，用于指定该方法可被远程调用。

例如，下面的示例代码使用“ServiceContract”属性声明接口 IMyFirstService 可以被远程调用，“OperationContract”属性声明 getTime()方法也可以被远程调用。

```
[ServiceContract]
public interface IMyFirstService
{
    [OperationContract]
    string getTime();
}
```

在表 13-4 中列出了 ServiceContract 属性的可用选项及其说明。

表 13-4　ServiceContract 属性的选项及说明

选　项	说　明
ConfigurationName	用于定义配置文件中服务配置的名称
CallbackContract	当服务用于双向消息传递时，此选项定义了客户端程序中实现的合约
Name	此选项定义了 WSDL 中 portType 节点的名称
Namespace	此选项定义了 WSDL 中 portType 节点的命名空间
SessionMode	此选项可定义调用这个合约的操作所需的会话。其值是 SessionMode 枚举值，可选的值有 Allowed、NotAllowed 和 Required
ProtectionLevel	此选项确定了绑定是否必须能保护通信。其值是 ProtectionLevel 枚举值，可选的值有 None、Sign 和 EntryptAndSign

在表 13-5 中列出了 OperationContract 属性的可用选项及其说明。

表 13-5　OperationContract 属性的选项及说明

选　项	说　明
Action	WCF 使用 SOAP 请求的 Action 选项把该请求映射到相应的方法上。因此使用此选项可以对请求的方法进行重命名
ReplyAction	此选项用于设置回应消息的 Action 名称
AsyncPattern	如果使用异步模式来实现操作，则可以将此选项设置为 true
IsInitiating	如果合约由一系列操作组成，则可以使用此选项指定初始化时执行的方法
IsTerminating	如果合约由一系列操作组成，则可以使用此选项指定结束时执行的方法
IsOneWay	使用此选项后，客户端程序将不会等待回应消息。因此在发送请求消息后，单向操作的调用者就无法直接检查是否失败
Name	操作的默认名称是方法的名称，使用此选项可进行重命名
ProtectionLevel	此选项可用于确定消息仅仅是签名，还是应先加密后签名



在服务合约中，还可以使用 DeliveryRequirements 属性定义服务的传输要求；使用 RequireOrderedDelivery 属性指定所传递的消息必须以相同的顺序到达；使用 QueuedDeliveryRequirements 属性指定消息以断开连接的方式传送。

2. 数据合约

数据合约也分为两种：DataContract 和 DataMember。DataContract 用于类或者结构上，指定此类或者接口能够被序列化并传输，而 DataMember 只能用在类或者接口的属性(Property)或者字段(Field)上，指定该属性或者字段能够被序列化传输。

数据合约的序列化不同于普通.NET 的序列化机制，在运行时所有的字段(包括私有字段)都会被序列化，而在执行数据合约的序列化时只有被标记了 DataMember 的属性才会被序列化。

例如，下面创建一个 Person 类并使用 DataContract 属性指定为可序列化。另外还创建了一个服务合约并定义了一个 Add()方法接受一个 Person 类型的参数。

```
[DataContract]
public class Person
{
    [DataMember]
    public int Id;
    [DataMember]
    public string Name;
    [DataMember]
    public DateTime Birthday;
    [DataMember]
    public string Email;
}
[ServiceContract]
public interface IPerson
{
    [OperationContract]
    bool Add(Person p);
}
```

在表 13-6 中列出了 DataMember 属性可用的选项及其说明。

表 13-6　DataMember 属性的选项及说明

选　项	说　明
Name	序列化时默认名称与类中的声明相同，使用此选项可以进行重命名
Order	获取或设置成员的序列化和反序列化的顺序
IsRequired	获取或设置一个值，该值用于指定序列化引擎在读取或反序列化时成员必须存在
EmitDefaultValue	获取或设置一个值，该值指定是否对正在被序列化的字段或属性的默认值进行序列化。默认值为 true

例如，我们要对 Order 对象进行序列化，它的定义如下。在这里我们使用 Namespace 选项重新定义了命名空间。对于数据成员使用 Name 选项指定了一个别名，使用 Order 选项指定了显示的顺序。

```
[DataContract(Namespace = "http://www.itzcn.com")]
public class Order
{
    [DataMember(Name="OrderId",Order=1)]
    public Guid ID;
    [DataMember(Name="OrderDate",Order=2)]
    public DateTime Date;
    [DataMember]
    public string Customer;
    [DataMember]
    public string Address;
    public double TotalPrice;
}
```

执行后，Order 对象的序列化 XML 如下所示：

```
<Order xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.itzcn.com">
   <Address></Address>
   <Customer></Customer>
   <OrderId></OrderId>
   <OrderDate></OrderDate>
</Order>
```

通过定义的数据合约以及与最终生成 XML 结构的对比，我们可以总结出 WCF 默认采用如下的数据合约序列化规则：

- XML 的根节点为数据合约中类的名称，默认命名空间格式为“http://schemas.datacontract.org/2004/07/{类所在命名空间}”。
- 只有使用 DataMember 属性定义的字段或者属性才能作为数据成员参与序列化(例如本实例中的 TotalPrice 属性不会再现在序列化后的 XML 中)。
- 所有数据成员均以 XML 元素的形式被序列化。
- 默认数据成员按照字母顺序排列。
- 如果通过 Order 指定了顺序，且值相同，则以字母先后顺序排列。
- 未指定 Order 的成员顺序在指定 Order 顺序之前。
- 如果 DataContract 处于继承的类中，那么将优先显示父类中的成员。

3．消息合约

如果需要在 WCF 服务中对 SOAP 消息进行控制则必须使用消息合约。在消息合约中，可以指定消息的哪些部分出现在 SOAP 标题中，哪一部分要放在 SOAP 的主体中。

例如，下面的示例演示了使用 ProcessOrderMessage 类定义消息合约的代码。

```
[MessageContract]
public class ProcessOrderMessage
{
   [MessageHeader]
   public Guid ID;
   [MessageBodyMember]
   public Order order;
}
```

如上述代码所示，在这里使用“MessageContract”属性指定 ProcessOrderMessage 为一个消息合约，对于 SOAP 消息中的标题使用“MessageHeader”属性指定，SOAP 主体使用“MessageBodyMember”属性指定。

为了使用上面定义的消息合约，我们将 IProcessOrder 接口定义为服务合约，并定义了一个可调用的 ProcessOrder()方法。代码如下所示：

```
[ServiceContract]
public interface IProcessOrder
{
   [OperationContract]
```

```
    ProcessOrderMessage ProcessOrder(ProcessOrderMessage message);
}
```

4. 异常合约

在 WCF 中所有的合约基本上都是围绕着一个服务调用时的消息交换来进行的。例如，服务的客户端向服务的提供者发送请求消息；服务提供者在接收到该请求后激活服务实例，并调用相应的服务操作；最终将返回的结果以回复消息的方式返回给服务的客户端。

但是，如果服务操作不能正确地执行，服务端将会通过一种特殊的消息将错误信息返回给客户端，这种消息被称为异常消息。对于异常消息，同样需要相应的合约来定义其结构，我们把这种合约称为异常合约(Fault Contract)。

WCF 通过 FaultContract 属性来定义异常合约。由于异常合约是基于服务操作级别的，所以该属性将直接应用于服务合约接口或者操作合约的方法上。

下面的示例代码演示了 FaultContract 定义异常合约的方式。

```
[ServiceContract]
public interface IUser
{
    [OperationContract]
    [FaultContract(typeof(LoginTimeOut))]
    bool Login(string username, string userpass);
}
```

在上述代码中，使用 FaultContract 属性声明调用 Login()方法会抛出 LoginTimeOut 类的异常，表示登录超时。

与本节前面介绍的其他合约一样，异常合约的 FaultContract 属性也有很多可用选项，如表 13-7 中列出了它们及其说明。

表 13-7　FaultContract 属性的选项及说明

选　项	说　明
Action	此选项用于设置当操作合约出现 SOAP 异常消息时要调用的操作。默认值为“当前操作的名称+Fault”
DetailType	此选项用于指定封装异常信息的自定义类，例如在上面定义的登录超时类 LoginTimeOut
Name	此选项用于设置 WSDL 中异常消息的名称
Namespace	此选项用于设置 SOAP 异常的命名空间
HasProtectionLevel	此选项用于设置 SOAP 异常消息是否分配有保护级别
ProtectionLevel	此选项用于设置 SOAP 异常消息要绑定的保护级别

13.4　配置 WCF 端点

上一节我们对 WCF 服务内部的核心概念进行了详细讲解。本节将从另一个角度来介绍

WCF，那就是端点。WCF 端点由上节介绍的地址、绑定和合约组成，它定义了 WCF 服务如何进行通信，以及使用什么地址进行通信。

 **视频教学：** 光盘/videos/13/配置 WCF 端点.avi　　　　长度：5 分钟

每个 WCF 服务都会关联到一个用于定位服务位置的地址，一个用于定义如何与服务进行通信的绑定，以及一个告知客户端服务能做什么的合约。这三样共同组成了服务的端点。

每个端点都必须完整拥有这三个组成部分，主机通过公开端点来对外提供服务。理论上，端点就是服务的外部交互接口，就像 CLR 或者 COM 接口。每个服务至少需要公开一个端点，服务上所有的端点都必须拥有唯一的定位地址，单个服务可以提供多个端点供不同类型的客户端调用。这些端点可以使用相同或不同的绑定对象，可以拥有相同或不同的服务契约。但是对于单个服务的不同端点，它们之间没有任何关联。

下面对如何通过配置文件和编程方式定义端点进行介绍。

1．通过配置文件方式

这种方式是将端点的信息保存到主机的配置文件中，通常是 app.config 文件或者 web.config 文件。

假设，我们使用如下代码定义了一个服务合约及其实现类。

```
//指定命名空间
namespace MyNameSpace
{
    [ServiceContract]
    public interface IMyContract
    {
        //这里是 IMyContract 接口的定义
    }
    public class MyService : IMyContract
    {
        //这里是 IMyContract 接口的实现
    }
}
```

如下给出了针对这个 WCF 服务的端点信息，其中包含服务的完整名称、绑定类型、合约的完整名称等。

```
<system.serviceModel>
    <services>
        <service name = "MyNameSpace.MyService">
            <endpoint
                address  = "http://localhost:8000/MyService/"
                binding  = "wsHttpBinding"
                contract = "MyNameSpace.IMyContract"
            />
        </service>
    </services>
</system.serviceModel>
```


注意

在这里指定的服务名称和服务合约必须是带命名空间的完整名称，否则将无法正确引用其地址。而且地址的类型与绑定类型必须匹配，否则就会在加载服务时导致异常。

当然，我们可以在配置文件中为一个单独的服务提供多个端点设置。这些端点可以使用相同的绑定类型，但是必须保证 URL 是唯一的。例如，如下是多个端点的示例配置文件：

```
<service name = "MyService">
    <endpoint
        address  = "http://localhost:8000/MyService/"
        binding  = "wsHttpBinding"
        contract = "IMyContract"
    />
    <endpoint
        address  = "net.tcp://localhost:8001/MyService/"
        binding  = "netTcpBinding"
        contract = "IMyContract"
    />
    <endpoint
        address  = "net.tcp://localhost:8002/MyService/"
        binding  = "netTcpBinding"
        contract = "IMyOtherContract"
    />
</service>
```

我们还可以提供一个或多个默认的基本地址(Base Address)，这样在端点设置中只需提供相对地址。多个基本地址之间不能冲突，不能在同一个端口进行监听。

相对地址通过端点绑定类型与基本地址进行匹配，从而在运行时获得完整地址。如果将某个端点设置中的地址设为空值(省略 address)，则表示直接使用某个相匹配的基本地址。

例如，如下的配置文件演示了这种方式：

```
<service name = "MyService">
    <host>
        <baseAddresses>
            <add baseAddress="http://localhost:8080/" />
            <add baseAddress="net.tcp://localhost:8081/" />
        </baseAddresses>
        </host>
        <endpoint
        address  = "MyService"       <!-- http://localhost:8080/MyService -->
        binding  = "wsHttpBinding"
        contract = "IMyContract"
    />
    <endpoint
        address  = "MyService"   <!-- net.tcp://localhost:8081/MyService -->
        binding  = "netTcpBinding"
        contract = "IMyContract"
    />
```

```
    <endpoint
        address  = "net.tcp://localhost:8002/MyService/"
        binding  = "netTcpBinding"
        contract = "IMyOtherContract"
    />
</service>
```

此外，还可以进一步对端点中的绑定参数进行设置。每种绑定类型可拥有多个名称不同的参数设置，然后在端点的 bindingConfiguration 属性中指定关联设置名称。

例如，在下面的配置文件中使用 bindingConfiguration 属性指定关联名称为 TransactionalTCP 的绑定信息。

```
<system.serviceModel>
    <services>
        <service name = "MyService">
            <endpoint
                address  = "net.tcp://localhost:8000/MyService/"
                bindingConfiguration = "TransactionalTCP"
                binding  = "netTcpBinding"
                contract = "IMyContract"
            />
        </service>
    </services>
    <bindings>
        <netTcpBinding>
            <binding name = "TransactionalTCP"
                transactionFlow = "true"
            />
        </netTcpBinding>
    </bindings>
</system.serviceModel>
```

2．通过编程方式

编程方式配置端点与使用配置文件的方式是等效的。它的优点是不需要编写额外的配置文件，而是通过编程的方式将端点添加到 ServiceHost 实例中。如下所示为创建 ServiceHost 实例时可用的两个构造函数形式：

```
public ServiceHost(object singletonInstance, params Uri[] baseAddresses);
public ServiceHost(Type serviceType, params Uri[] baseAddresses);
```

ServiceHost 提供了一个 AddServiceEndpoint()方法向当前的服务中添加端点。如下所示为该方法的重载形式：

```
public ServiceEndpoint AddServiceEndpoint(Type implementedContract, Binding
binding, string address);
public ServiceEndpoint AddServiceEndpoint(Type implementedContract, Binding
binding, Uri address);
public ServiceEndpoint AddServiceEndpoint(Type implementedContract, Binding
binding, string address, Uri listenUri);
```

```
public ServiceEndpoint AddServiceEndpoint(Type implementedContract, Binding
binding, Uri address, Uri listenUri);
```

在使用时 address 参数可以是相对地址，也可以是绝对地址，这与使用配置文件是一致的。例如，下面的代码演示了如何通过编程方式配置端点：

```
ServiceHost host = new ServiceHost(typeof(MyService));
Binding wsBinding  = new WSHttpBinding( );
Binding tcpBinding = new NetTcpBinding( );
host.AddServiceEndpoint(typeof(IMyContract), wsBinding,
"http://localhost:8000/MyService");
host.AddServiceEndpoint(typeof(IMyContract), tcpBinding,
"net.tcp://localhost:8001/MyService");
host.AddServiceEndpoint(typeof(IMyOtherContract), tcpBinding,
"net.tcp://localhost:8002/MyService");
host.Open( );
```

13.5　创建 WCF 服务主机

我们已经看到，在 WCF 中客户端和服务端是通过端点来通信的。而 WCF 服务主机实际就是把一个服务置于一个运行的进程中，然后再向客户端公开可调用的端点，并监听来自客户端的请求。

**视频教学：**光盘/videos/13/WCF 服务主机.avi　　长度：2 分钟

在 WCF 中指定 WCF 的启动主机时有很多选择。主机可以是 Windows 服务、COM+应用程序、ASP.NET 应用程序、Windows 应用程序或者控制台应用程序。

例如，下面给出的示例代码使用控制台应用程序来启用 WCF 主机。

```
static void Main(string[] args)
{
    using (ServiceHost host = new ServiceHost())
    {
        host.Open();
        Console.Write("服务已经启动，按任意键结束服务！");
        Console.Read();
        host.Close();
    }
}
```

可以看到，在 Main()方法中创建了一个 ServiceHost 实例。接着，读取应用程序配置文件来定义绑定，也可以像前面介绍的那样通过编程定义绑定。再往下调用 ServiceHost 实例的 Open()方法使服务可以接受客户端的调用。最后，在使用完成后调用 Close()方法结束服务。



如果要中止 WCF 服务主机，可以调用 ServiceHost 实例的 Abort()方法。还可以使用它的 State 属性获取当前服务的状态。

13.6　实现除法运算的 WCF 服务

学到此处，相信读者一定对 WCF 有了更深的了解，而且应该能够根据项目的需求合理设计 WCF 服务了。例如，根据网络情况选择一种通信协议、设计一种绑定方式以及使用合约指定可被调用的方法。

本节将通过一个完整的实例总结 WCF 服务的应用。实例运行在控制台下，实现了除法运算并可以对除数为零的情况进行捕捉。

**视频教学：**光盘/videos/13/除法运算 WCF 服务.avi　　长度：8 分钟

13.6.1　实例描述

本实例使用 WCF 服务库项目作为服务器，一个控制台程序作为客户端。然后客户端通过服务引用建立对 WCF 的连接，再调用 WCF 提供的方法实现除法运算。整个实例的制作过程都在 Visual Studio 2010 中完成。

13.6.2　实例应用

【例 13-2】 实现除法运算的 WCF 服务

(1) 在 Visual Studio 2010 中新建一个名为“CalculatorWcfServiceLibrary”的 WCF 服务库项目。

(2) 按照 WCF 的开发流程，先定义服务再实现，最后调用。这里向项目中添加一个名为 ICalculator.cs 的接口。

(3) 在接口中声明一个用于执行除法运算的 Divide()方法，然后将接口和方法设置为可被远程调用，代码如下所示：

```
[ServiceContract(Namespace = "http://www.itzcn.com")]
public interface ICalculator
{
    [OperationContract]
    [FaultContract(typeof(MathError))]
    double Divide(double x, double y);
}
```

(4) 在上述代码中使用 FaultContract 属性定义了一个 MathError 类型的异常合约。这一步我们创建这个 MathError 类，由于它将在 WCF 服务中使用，因此又使用 DataContract 属性设置为数据合约。具体实现代码如下所示：

```
[DataContract]
public class MathError
{
    public MathError(string operation, string errorMessage)
    {
        this.Operation = operation;
        this.ErrorMessage = errorMessage;
    }
    [DataMember]
    public string Operation { get; set; }
    [DataMember]
    public string ErrorMessage { get; set; }
}
```

(5) 下面来编写接口的实现代码。在项目中添加一个名为 CalculatorService 的类，使它实现 ICalculator 接口。CalculatorService 类的完整代码如下所示：

```
public class CalculatorService : ICalculator
{
    public double Divide(double x, double y)
    {
        if (y == 0)
        {
            MathError error = new MathError("除法", "被除数不能为零。");
            throw new FaultException<MathError>(error, new FaultReason
                ("参数传递无效。"), new FaultCode("sender"));
        }
        return x / y;
    }
}
```

(6) 完成 CalculatorService 类的代码后，打开项目中的 App.config 文件。在这里配置上面设计服务的端点信息，最终内容如下所示：

```
<services>
    <service name="CalculatorWcfServiceLibrary.CalculatorService">
        <endpoint address="" binding="wsHttpBinding" contract="
            CalculatorWcfServiceLibrary.ICalculator">
            <identity>
                <dns value="localhost" />
            </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="
            IMetadataExchange" />
        <host>
            <baseAddresses>
                <add baseAddress="http://localhost:8732/
                CalculatorWcfServiceLibrary/CalculatorService/" />
            </baseAddresses>
        </host>
    </service>
</services>
```

(7) 到这一步针对 WCF 项目的工作就完成了，我们可以直接运行测试。这里我们使用控制台进行测试，所以向解决方案中添加一个名为"TestCalculatorConsoleApplication"的控制台应用程序。

(8) 右击项目选择【添加服务引用】命令，打开【添加服务引用】对话框。在这里单击【发现】按钮来查找本解决方案中的服务。找到后会看到服务的 URL 地址，以及可调用的操作。在下方的【命名空间】文本框中设置值为"ServiceReference1"，最后单击【确定】按钮完成引用，如图 13-13 所示。

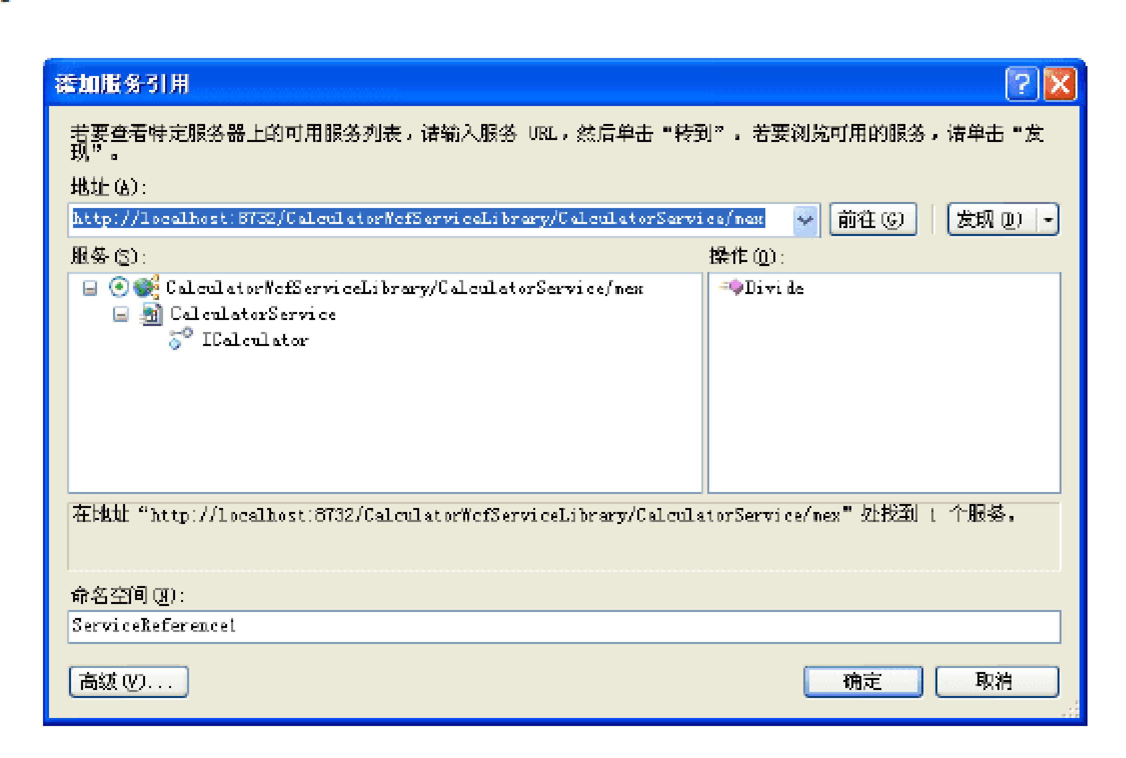


图 13-13　【添加服务引用】对话框

(9) 在控制台程序中添加对如下命名空间的引用。

```
using System.ServiceModel;
using TestCalculatorConsoleApplication.ServiceReference1;
```

(10) 在 Main()方法中编写具体的实现代码，包括实例化 WCF 服务类，调用方法，输出结果以及处理异常等。完整代码如下所示：

```
static void Main(string[] args)
{
    ServiceReference1.CalculatorClient Calc = new ServiceReference1.
        CalculatorClient();
    try
    {
        Console.WriteLine("执行第一次除法运算");
        Console.WriteLine("x / y =  {2} when x = {0} and y = {1}", 4, 2,
            Calc.Divide(4, 2));
        Console.WriteLine();
        Console.WriteLine("执行第二次除法运算");
        Console.WriteLine("x / y =  {2} when x = {0} and y = {1}", 9, 2,
            Calc.Divide(9, 2));
        Console.WriteLine();
        Console.WriteLine("执行第三次除法运算");
        Console.WriteLine("x / y =  {2} when x = {0} and y = {1}", 9, 0,
            Calc.Divide(9, 0));
        Console.WriteLine();
    }
    catch (FaultException<MathError> ex)
```

```
    {
        MathError error = ex.Detail;
        Console.WriteLine("An Fault is thrown.\n\tFault code:{0}\n\
            tFault Reason:{1}\n\tOperation:{2}\n\tMessage:{3}", ex.Code,
            ex.Reason, error.Operation, error.ErrorMessage);
    }
    catch (Exception ex)
    {
        Console.WriteLine("An Exception is thrown.\n\tException Type:{0}
            \n\tError Message:{1}", ex.GetType(), ex.Message);
    }
    Console.WriteLine(); Console.WriteLine("按任意键退出程序!");
    Console.ReadKey();
}
```

(11) 右击解决方案选择【生成解决方案】命令，生成上面创建的项目，完成实例的制作。

13.6.3　运行结果

在控制台程序中按 F5 键执行，此时 Visual Studio 2010 将在“http://localhost:8732/CalculatorWcfServiceLibrary/CalculatorService”位置启动 WCF 服务主机等待调用。

待执行到除法运算时，由于设置了被除数为零的情况，所以首先会看到如图 13-14 所示的提示。



图 13-14　被除数为零时的异常提示

按 F5 键继续运行，此异常将会被 WCF 服务的调用方捕捉到，并看到如图 13-15 所示的运行效果。

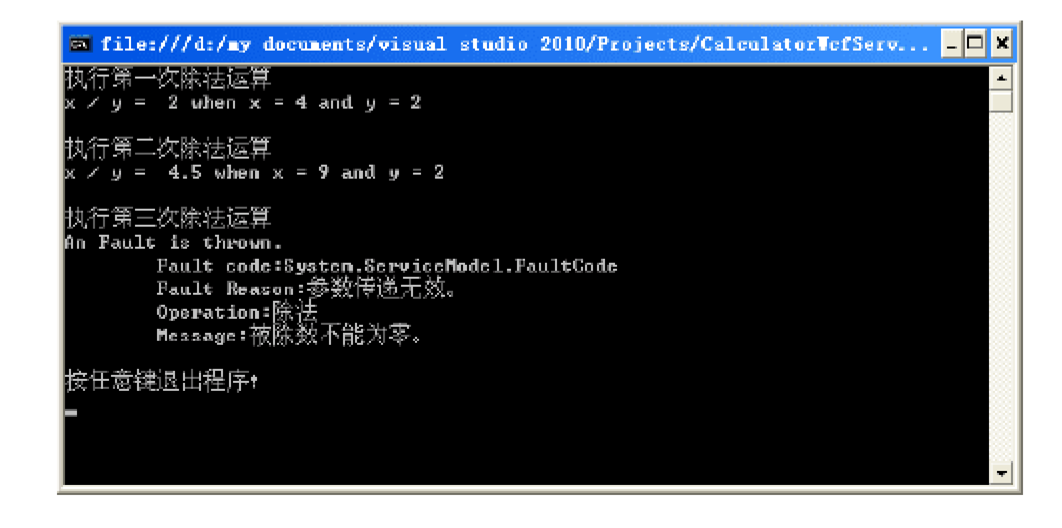


图 13-15　运行效果

13.6.4　实例分析



源码解析：

在实例中演示了 WCF 服务的另一个引用方式，即通过添加服务引用来完成。这种方式省掉了手动执行命令、生成代理和添加文件等复杂的步骤。从而使开发人员可以更专心地编写 WCF 服务的业务逻辑以及实现。

本实例实现的功能虽然很简单，但是它用到了本章介绍的所有 WCF 知识。最重要的是使读者掌握 WCF 服务应用程序的开发流程。

最后要注意，WCF 的很多功能都依赖于 System.ServiceModel 命名空间。因此，在设计时第一步就是添加对它的引用。

13.7　常见问题解答

13.7.1　WCF 中的合约和哪种技术比较类似



WCF 中的合约和哪种技术比较类似？
网络课堂：http://bbs.itzcn.com/thread-15741-1-1.html

向各位大侠请教一下，WCF 中的合约和哪种技术比较相像，又有什么不同呢？

【解决办法】

如果非要拿合约和以往的技术相比较。合约和 ASP.NET Web Service 的声明性编程模型非常相似。例如，在 Web Service 中的类上标记 WebService 属性便可以将此类用于远程调用，而将方法添加 WebMethond 属性也可以将其暴露给远程客户端。这和 WCF 中的 ServiceContract 及 OperationContract 简直如出一辙。

但不同的是，WCF 中的合约要比 WebService 中的详尽，例如，ServiceContract 和 OperationContract 可以直接使用在接口上面，而实现该接口的类就继承了这种合约声明。自动拥有合约所规范的动作和行为，这就使得程序员更方便地使用面向接口的编程方式。从而可以使同一服务拥有不同的实现，能够使新老版本共同运行。

13.7.2　菜鸟请教一个 WCF 问题



菜鸟请教一个 WCF 问题？
网络课堂：http://bbs.itzcn.com/thread-15742-1-1.html

我创建了一个 WcfServiceLibrary1 和一个 ConsoleApplication1，然后在 ConsoleApplication1 里添加服务引用。请问以下几个问题：

(1) 那么是不是说 ConsoleApplication1 就是客户端，WcfServiceLibrary1 就是服务端？

(2) 如果是的话，在客户端“ServiceHost host = new ServiceHost(typeof(UserService)); host.Open();”启动服务后，怎么调用服务端的方法呢？

(3) 还有这样一种写法“IService1 s = new Service1Client();serv.GetData(23)”。这又是什么意思呢？

【解决办法】

WCF 的服务器端一般分成三个部分。第一个是 Contract，这部分和 Client 应该是公用同一个 dll 文件，用于定义数据规范。第二个是主机程序，可以是 IIS，可以是 Console 或者其他类型程序，用来加载和开启服务，一直监听并等待 Client 的请求。第三个是提供 Service 的 dll，也就是实现 Contract 中定义的 Service。

Client 方面，除了 Contract，就是和服务器交互的那个类。

根据你的描述，应该是 WcfServiceLibrary1 用于提供 Service 接口，ConsoleApplication1 程序用于启用 WCF 服务。那么在该程序中会用到“ServiceHost host = new ServiceHost(typeof (UserService)); host.Open();”代码启动 WCF 服务。

13.7.3 WCF 使用时的问题



WCF 使用时的问题？

网络课堂：http://bbs.itzcn.com/thread-15743-1-1.html

我用 Visual Studio 2010 创建了一个 WCF 的服务库，然后创建了客户端，并添加服务引用，此时生成了一个配置文件。请问以下几个问题。

(1) 客户端生成的配置文件是根据服务端的设置生成的吗？客户端还需要在这基础上配置什么？

(2) 网上说用微软自带的什么命令工具生成客户端配置文件，生成的是什么配置文件啊？

(3) WCF 能用 UDP 协议传输图片文件吗？

(4) WCF 客户端怎么向服务器发送文本和图片呢(不用 socket 传输)？

【解决办法】

(1) 添加引用的时候是根据地址找到服务的，所以生成的客户端配置文件自然要包含一些服务器端的信息。因此，可以说是根据服务器端的设置生成的。如果你是用 Visual Studio 2010 添加服务引用的，基本上不用其他的配置了。

(2) 那是生成客户端代理的工具，命令为 svcutil.exe。它生成客户端代理的同时会同时生成一个配置文件，在客户端用。

(3) WCF 中有个 binding 的概念。binding 描述了服务传输的通信方式，使用绑定可以指定传输协议、安全要求、编码方式、事务处理要求和可靠性等。另外，WCF 的 binding 没有基于 UDP 协议的绑定，主要以 TCP 和 HTTP 为主。

(4) 具体的实现过程没有试过，但是应该不是问题。基本思路：文件→流→根据 WCF 中文件传输的方法进行文件流传输→转化为目的文件。

13.8 习 题

一、填空题

(1) WCF 是由__________开始引入，主要用于处理进程之间消息的传递。

(2) 在一个 WCF 服务中__________部分决定了服务的运行环境。

(3) 每个 WCF 端点都由绑定、__________和合约组成，其中__________指定如何与外界通信。

(4) 为了创建一个 WCF 服务，需要在类中使用__________属性声明服务合约，使用__________属性声明操作合约。

(5) WCF 支持的传输方式有 Peer Network、__________、TCP、IPC 和__________。

(6) 对于一个 WCF 主机，可以调用 ServiceHost 实例的__________方法启动服务，__________方法中止服务，__________方法结束服务。

二、选择题

(1) 下面关于 WCF 的描述，不正确的是__________。

A. WCF 是一个面向 Windows 编程的分布式架构

B. WCF 集成了 DCOM、Enterprise Service 以及 Web Service

C. 使用 WCF 可以对服务进行定制、发布与运行

D. WCF 解决了跨平台特性

(2) 在 WCF 中使用__________来定义访问地址。

A. Contract　　B. Binding　　C. Address　　D. EndPoint

(3) WCF 的合约必须要以__________的方式定义。

A. 类　　B. 接口　　C. 结构　　D. 数组

(4) 为了能够创建和使用 WCF，必须在项目中引用命名空间__________。

A. System.ServiceModel

B. System.Web.WCF

C. System.WCF

D. System.Net.WCF

(5) 对于一个 WCF 地址，下面__________选项是错误的。

A. http://localhost:2156/services/wcf

B. net.pipe:// localhost:2156/services/wcf

C. ip://192.168.0.135:2156/services/wcf

D. net.tcp:// localhost:2156/services/wcf

(6) 下列给出的绑定类型中，__________类型不是使用二进制编码进行通信的。

A. NetMsmqBinding

B. NetTcpBinding

C. NetMsmqBinding

D. WSHttpBinding

(7) WCF 中 DataContract 属性的__________选项可以设置成员的序列化顺序。

A. Order B. IsRequired C. Action D. DetailType

三、上机练习

上机练习 1: 创建一个简单的 WCF 服务应用程序。

本章详细介绍了 WCF 服务的核心概念、组成部分以及 WCF 服务的编写和调用方法。在本次上机练习中要求读者根据下面的要求创建一个 WCF 服务，并编写相应的配置文件、ASP.NET 客户端程序，并最后在浏览器中查看结果。

(1) WCF 服务中包含一个名为“IService1”的接口，由它实现服务合约。

(2) 使用如下代码声明一个返回字符串类型的 FormatDateTime()方法，它根据 format 参数格式化当前时间。

```
[OperationContract]
string FormatDateTime(string format);
```

(3) 使用如下代码声明一个返回浮点类型的 GetTemperature()方法，它根据 zipCode 参数返回指定城市的当前温度。

```
[OperationContract]
float GetTempearture(string zipCode);
```

(4) 使用如下代码声明一个返回 City 类型的 GetWeatherInfoById()方法，它根据 cId 参数返回一个自定义的 City 类型。

```
[OperationContract]
[FaultContract(typeof(WeatherInfoError))]
City GetWeatherInfoById(string cId);
```

(5) 创建 GetWeatherInfoById()方法所需的 City 类和 WeatherInfoError 类。

(6) 创建一个名为“Service1”的类继承 IService1 接口，并编写实现代码。

(7) 修改 WCF 的配置文件，使用 HTTP 地址 http://localhost:8080/MyWcfService 进行绑定。

(8) 创建一个 ASP.NET 应用程序，添加对 WCF 的服务引用，并在页面显示输出结果。



第 14 章　网络聊天工具

内容摘要：

相信所有的读者都用过 QQ 之类的一些聊天工具吧，一定感觉非常不错！

对于身处软件开发行业的我们来说，也不只一次地梦想要开发一个类似 QQ 一样的聊天工具，以供所有人使用。

当然，使用.NET 实现一个简单的聊天功能的应用程序并不难，但是像他们这些最早的网络服务提供商来说，基本上使用的是直接操作网络流的方式(比如 Socket，网络应用程序套接字)。使用这些传统的方法执行网络信息传递非常费事，而且容易出问题。

Web 服务提供了一种很简便的操作服务器远程方法的机制，可以让我们像使用本地应用程序一样访问网络中的一些服务。

在 ASP.NET 中，因为它对访问网络的操作进行了封装，所以我们可以使用 ASP.NET Web 服务很方便地调用远端方法，以实现服务器和客户端之间的数据传递。有了 Web 服务，我们距编写一个属于自己的聊天工具的梦想就越来越近了。

本章，我们就在 ASP.NET 中使用 Web 服务来开发一个简单的网络聊天工具。

学习目标：

- 掌握 Web 服务的创建方法
- 掌握使用应用程序调用 Web 服务的方法
- 掌握 Web 服务中用户状态保持的方法
- 掌握使用 Web 服务传递二进制文件的方法

14.1 系统需求和应用程序设计

一个聊天工具，最基本的需要是区分不同的聊天者，所以这里需要有不同的用户。

在这个聊天应用程序系统中会有非常多的用户，我们需要让用户有选择性地添加自己喜欢的一些用户与之聊天，所以这个系统中需要有“添加好友”这项功能。当然好友也不可能让你随意想添就添了，我们还需要实行相应的好友验证功能。

剩下的，就是最基本的聊天功能了。

当然，为了更加适用，我们还在该应用程序中添加了“传送文件”的功能，让聊天者可以在聊天的同时更加方便地使用文件来交流一些信息。



不过这里为应用程序的性能考虑，限制用户只能发送大小在 10KB 以下的文件。

14.1.1 系统需求

综合上面的这些需求，我们的应用程序需要包含以下几个功能。

- 用户注册：用户注册功能允许所有运行我们的客户端应用程序的用户执行，用户的注册信息通过 Web 服务提交到 Web 服务器中。注册成功的用户可以使用登录和聊天等其他功能。
- 用户登录：用户需要登录以后才能执行添加好友和聊天等功能。
- 添加好友：新用户注册以后，没有一个好友，我们需要在登录以后为该用户添加好友。添加好友操作需要由当前登录用户向对方发出请求，并等待对方验证，验证通过以后才算好友添加成功。这里要求不能向不存在的用户发出好友请求，不能向好友发出请求，当前用户向同一个用户发出好友请求以后，系统只接收一次请求，而对其他请求默认放弃。
- 好友验证：用户在接收到其他用户的好友请求以后，可以选择接受请求或者拒绝该用户请求。当用户接受某用户的好友请求以后，只说明当前用户自己成为那个用户的好友，而自己用户列表中并没有出现那个发出好友请求的用户。
- 发送消息：当前登录用户可以选中好友列表中的某一位好友，向其发送文本消息。如果用户没有选中任意一位好友，提示只能给好友发送消息；如果用户没有输入任何消息内容，则放弃当次消息的发送。
- 发送文件：用户可以向好友发送不大于 10KB 的任意类型的文件。当然，接受信息的用户可以选择接收或放弃接收该文件。

整个系统各功能的运行流程如图 14-1 所示。

新用户
注册
已注册用户
登录
添加好友
发送文件
发送消息
处理消息

图 14-1　系统各功能执行流程

14.1.2　应用程序设计

知道了应用程序的整个功能需求，我们就可以着手来设计应用程序了。

1．应用程序结构

该应用程序使用 C/S(Client/Server)结构的设计。服务器端采用 ASP.NET Web 服务作为整个系统的核心，也是数据中心。客户端使用.NET Windows Form 应用程序实现，请求调用 Web 服务来实现数据交换。

在服务器端使用三层架构，数据库使用 SQL Server。在数据访问层使用 LINQ to SQL 来封装操作数据库的方法，然后在业务逻辑层使用 LINQ 来操作数据对象取得数据，并处理应用程序业务。

如此，整个应用程序结构如图 14-2 所示。

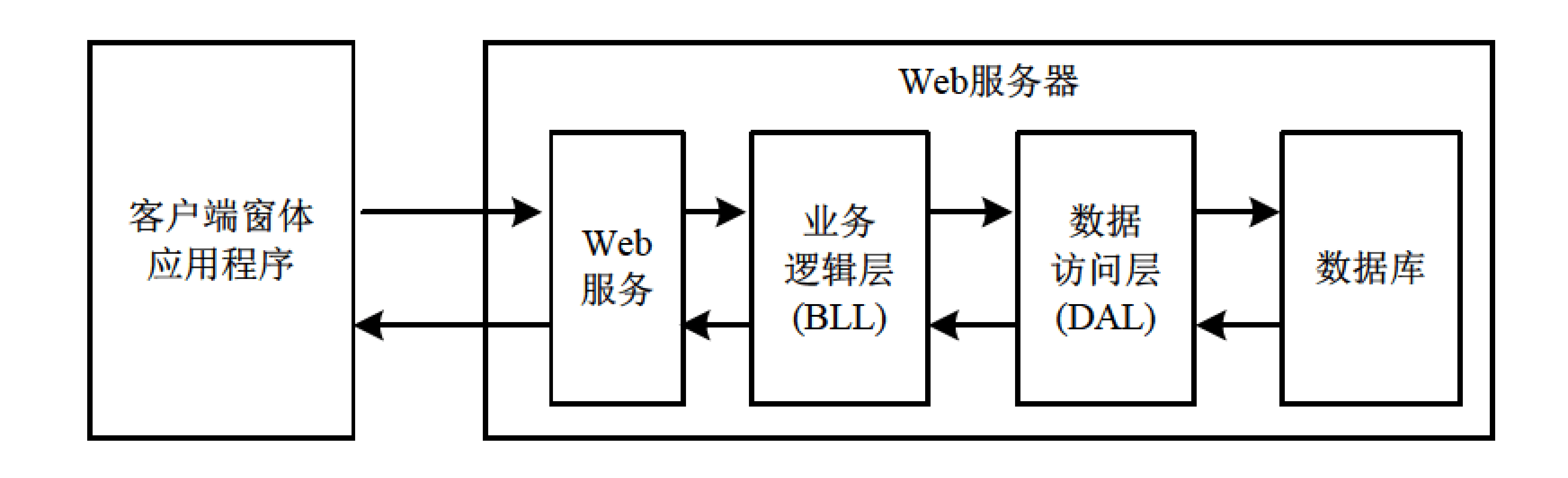


图 14-2　应用程序结构

2．数据库设计

在该系统中，我们需要抽出来几个名词：用户、好友、消息、文件、好友请求信息、好友响应信息。

在应用程序需求中，好友关系并不是双向的好友关系，而是可能你是我的好友，而我并不是你的好友(现在运行的几乎所有的聊天工具都是这么设计好友关系的)。所以我们不能单纯地建立好友关系表，要考虑单向的好友关系。在数据库中，我们就要设计任意一方的一对多关系映射。

而且，在好友关系中，并不是单纯地建立起来就行了，我们还需要面对好友请求，以及用户接不接受该请求的问题，所以我们要给好友关系添加一个是否接受请求的标识。

其次，消息、文件、好友请求信息、好友响应信息，这些东西都有一个共同的结构，即都有发送者、接收者，并且同属于一类东西——“信息”。所以我们可以将这些信息归结为一个实体对象，使用一个对象属性来区分不同的消息类型即可。



这样将所有的“信息”归为一类，使用标识符区分不同类别的方法也有利于以后的功能扩展，非常方便。

在所有“信息”对象中，消息不仅有发送者和接收者，还需要有消息内容；文件除了发送者和接收者以及文件内容外，还需要有一个文件名；而好友请求信息和好友响应信息没有必要具有信息内容，只需要区分开发送者和接收者就行了。另外，为了确认发送的信息的时效性，我们还需要为“信息”对象添加一个时间戳。

因此，我们的整个应用程序系统需要三个实体对象：用户、好友关系和信息。对于这三个实体对象的具体属性，我们可以分别做如下设计。

- 用户：用户昵称(同样也是登录名)、密码。
- 好友关系：用户名称、好友名称、是否验证通过。
- 信息：信息类型、发送者、接收者、信息内容、发送文件名(只对文件消息有效)、发送时间戳。

对于这三个实体对象在数据库中的结构，我们可以进行简单的设计，如表 14-1～表 14-3 所示。

表 14-1　用户表(Users)设计

字 段 名	说　明	数据类型	备　注
ID	编号	int	主键，自增
Nickname	昵称	varchar(20)	唯一约束
Password	密码	varchar(20)	

表 14-2　好友关系表(Friends)设计

字 段 名	说　明	数据类型	备　注
ID	编号	int	主键，自增
UserName	用户名	varchar(20)	Users 表 Nickname 字段外键
FriendName	好友用户名	varchar(20)	Users 表 Nickname 字段外键
IsPass	是否通过验证	bit	

表 14-3　信息表(Messages)设计

字 段 名	说　明	数据类型	备　注
ID	编号	int	主键，自增
Classify	分类	int	
Sender	发送者	varchar(20)	Users 表 Nickname 字段外键
Receiver	接收者	varchar(20)	Users 表 Nickname 字段外键
Details	详细内容	text	
FileFullName	文件名	varchar(50)	
SendTime	发送时间	datetime	

根据上面的设计，在 SQL Server 中创建数据库的 T-SQL 代码如下：

```
create table Users
(
    ID int identity(1,1) primary key,
    Nickname varchar(20) not null unique,
    Password varchar(20) not null
)

create table Friends
(
    ID int identity(1,1) primary key,
    UserName varchar(20) foreign key references Users(Nickname),
    FriendName varchar(20) foreign key references Users(Nickname),
    IsPass bit
)

create table Messages
(
    ID int identity(1,1) primary key,
    Classify int,
    Sender varchar(20) foreign key references Users(Nickname),
    Receiver varchar(20) foreign key references Users(Nickname),
    Details text,
    FileFullName varchar(50),
    SendTime datetime
)
```

3. 客户端界面设计

根据上面的需求，我们的应用程序服务器端使用 Web 服务来提供相应的接口，然后在客户端调用 Web 服务来实现应用程序的功能。

在客户端，使用.NET 的 Windows 窗体应用程序来创建用户界面。要实现需求中的所有功能，需要使用 4 个 Windows 窗体来完成操作，分别是注册窗体、登录窗体、聊天窗体和添加好友窗体。

注册窗体中，只需要接收用户输入的用户名和密码即可。用户密码非明文显示，所以需要用户确认密码操作。整个窗体只需要 3 个文本框接收用户输入即可，如图 14-3 所示。

用户登录窗体是整个应用程序运行中出现的第一个窗体，也是整个应用程序所有功能的入口点。我们在登录窗体中可以打开注册窗体，也可以登录到程序主界面。所以登录窗体可以设计成如图 14-4 所示的效果。

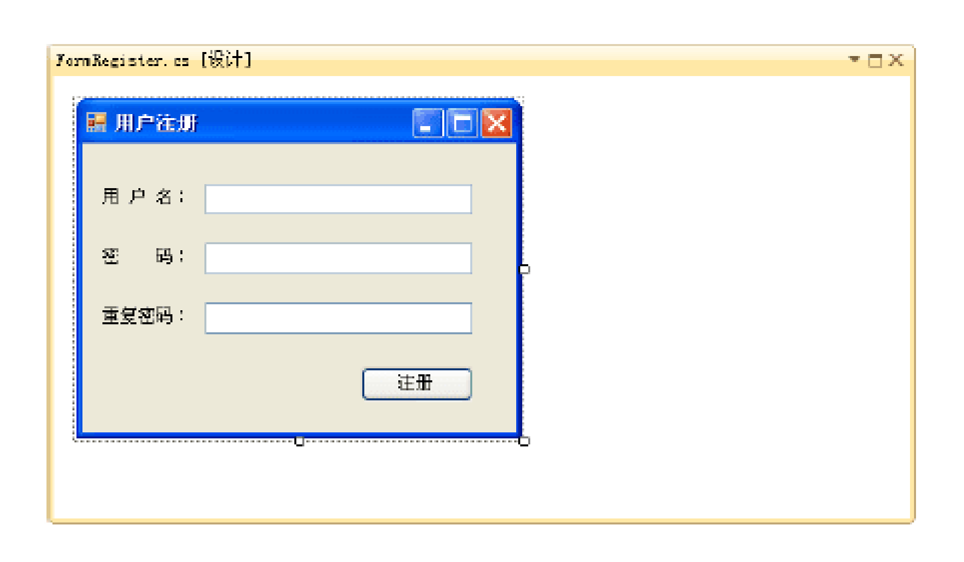


图 14-3 用户注册窗体

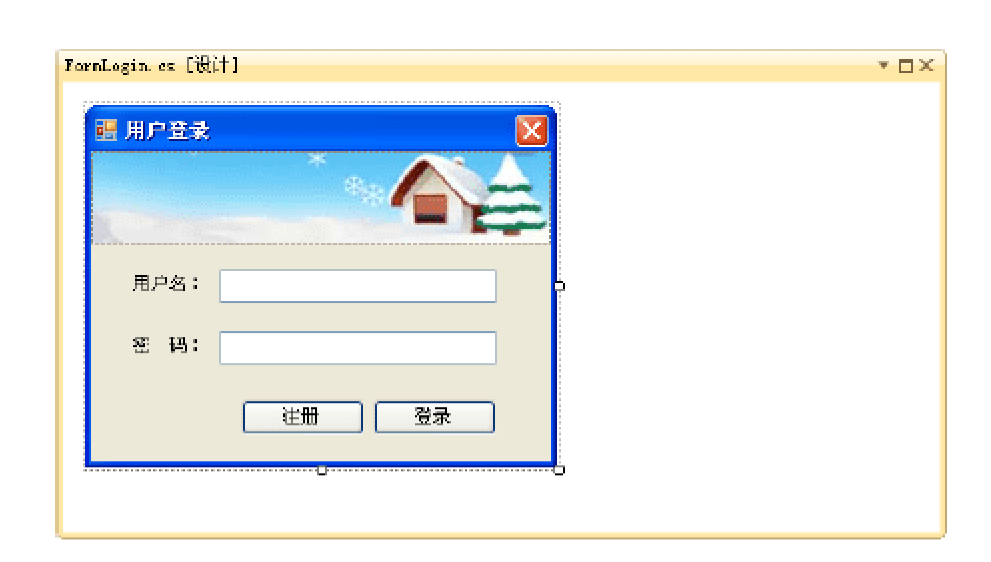


图 14-4 用户登录窗体

聊天窗体是整个应用程序的主窗体，在这里可以执行查看好友列表、刷新好友列表、发送消息、发送文件、打开等操作。设计效果如图 14-5 所示。

添加好友窗体相对来说非常简单，只需要接收用户输入的要添加好友的用户名，对其发送请求即可。设计效果如图 14-6 所示。

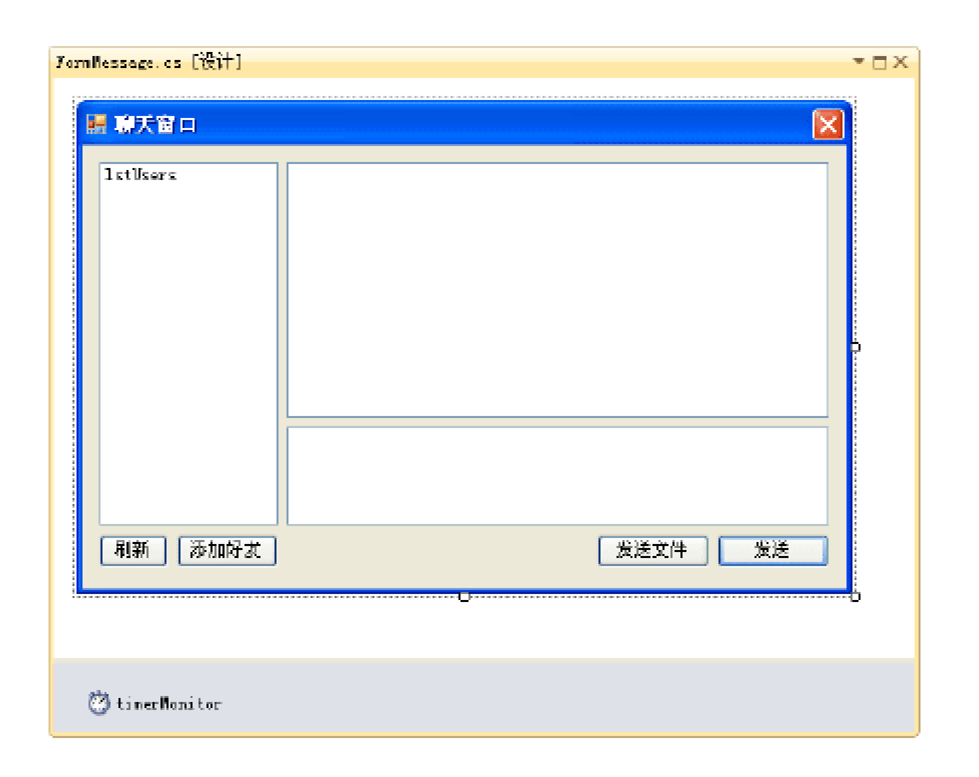


图 14-5 聊天窗体

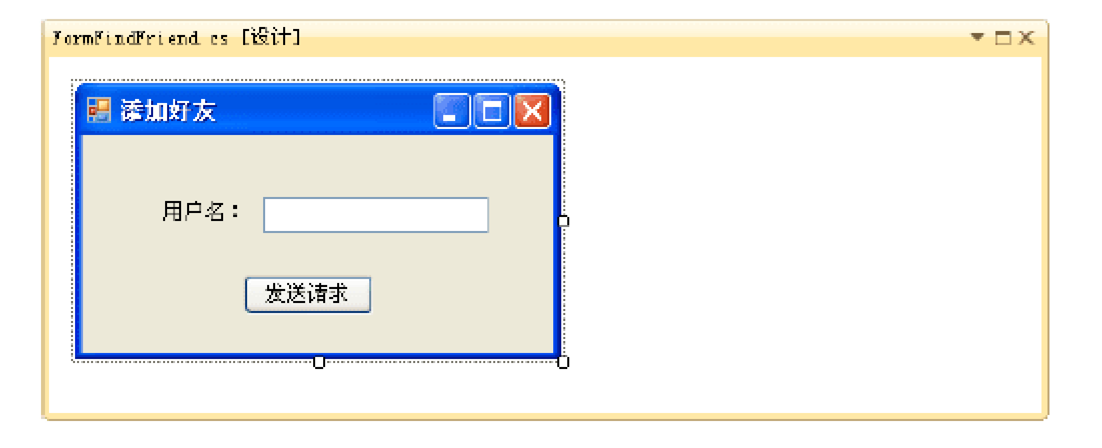


图 14-6 添加好友窗体

4．特殊功能设计

另外，在使用 Web 服务的应用程序中，Web 服务器不可能主动向客户端发送信息，所以我们需要将信息暂存在服务器中，然后等待用户请求。

而客户端正式启动以后需要定时不停地向服务器发出请求，反复地查询监听服务器中的消息。

14.2 服务器端设计

服务器端需要集中执行用户注册、验证用户存在、用户登录、添加好友、处理好友请求、发送消息、发送文件、监听信息、查询好友列表等功能。

14.2.1　创建项目结构

这里我们使用 Web 服务实现在互联网中发布的功能，所以需要使用一个 ASP.NET Web 服务项目。

当然我们这里需要操作数据库，并且需要处理一些业务逻辑，所以使用三层架构的模式来设计项目。

首先创建一个解决方案，命名为 WebServiceApp；在解决方案中创建一个用于发布 Web 服务的 Web 项目，也命名为 WebServiceApp；然后再创建一个用于封装数据访问操作的类库项目，命名为 Web.DAL；最后创建一个用于封装业务逻辑的类库项目，命名为 Web.BLL。

上面所有的工作执行完以后，Visual Studio 中的解决方案资源管理器中的效果如图 14-7 所示。

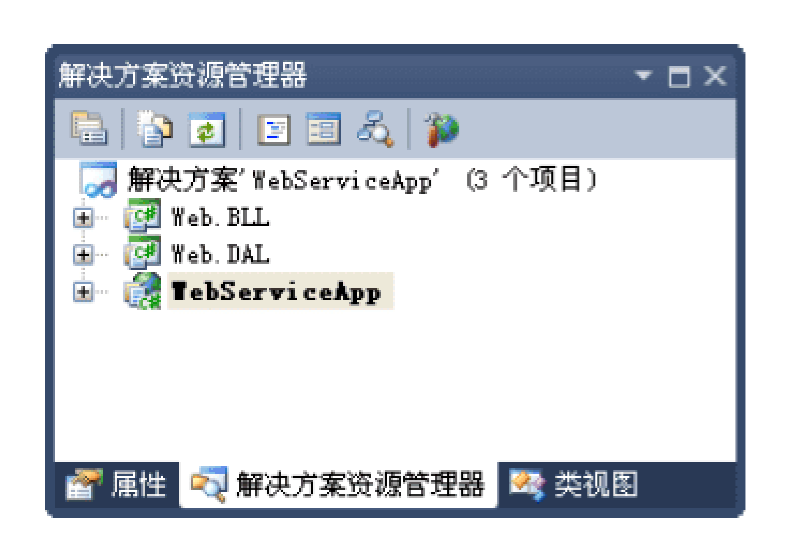


图 14-7　解决方案资源管理器

完成以后我们还要为这三个项目创建引用关系。这三个项目的引用关系为 Web.BLL 引用 Web.DAL，WebServiceApp 引用 Web.BLL 和 Web.DAL。

14.2.2　添加数据访问类

因为使用的是 SQL Server 数据库，所以在项目中使用 LINQ to SQL 来执行数据库操作是非常方便的事情。

这里我们在 Visual Studio 的 Web.DAL 项目中添加一个 LINQ to SQL 类，命名为 DataMessage。创建以后打开这个类。

然后在“服务器资源管理器”中添加到实例数据库的连接，并浏览该数据库中的表。



这里需要首先创建一个数据库，并使用前面的创建数据库表的代码创建需要的数据库表。

在“服务器资源管理器”中选中数据库表 Friends、Messages 和 Users，并用鼠标将其拖入创建的 LINQ to SQL 类 DataMessage 中。执行结果如图 14-8 所示。

通常来说，这里创建完成以后，就可以正常使用了。但是，由于 LINQ to SQL 类自动创建的模型对象会根据表之间的关联创建相互的引用关系，这在使用 XML 序列化数据对象的时候就会造成无限循环的引用。所以我们要进行一些设置，避免这种问题。

首先在 DataMessage 类的属性窗口中设置序列化模式为“单向”，如图 14-9 所示。

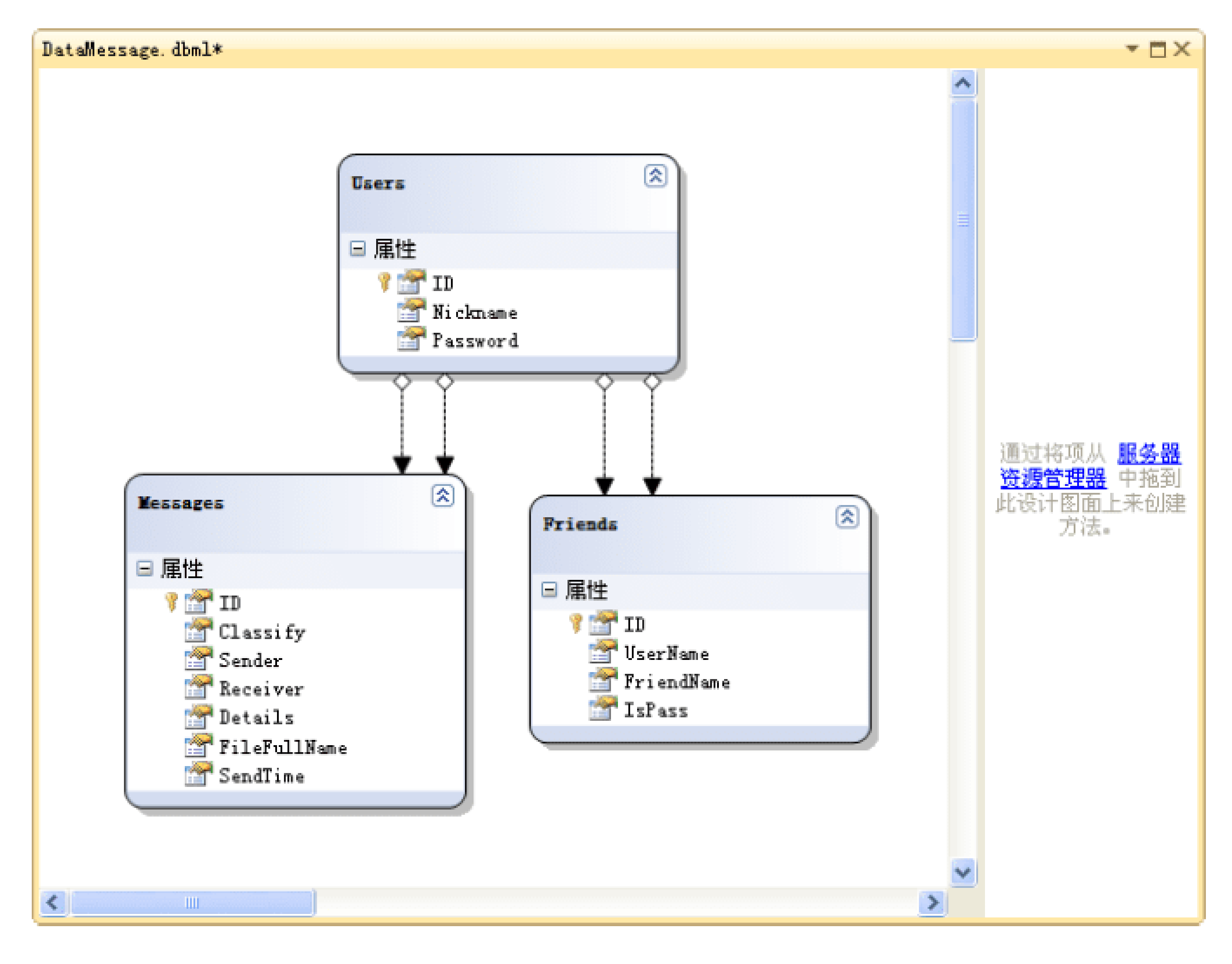


图 14-8　DataMessage 类

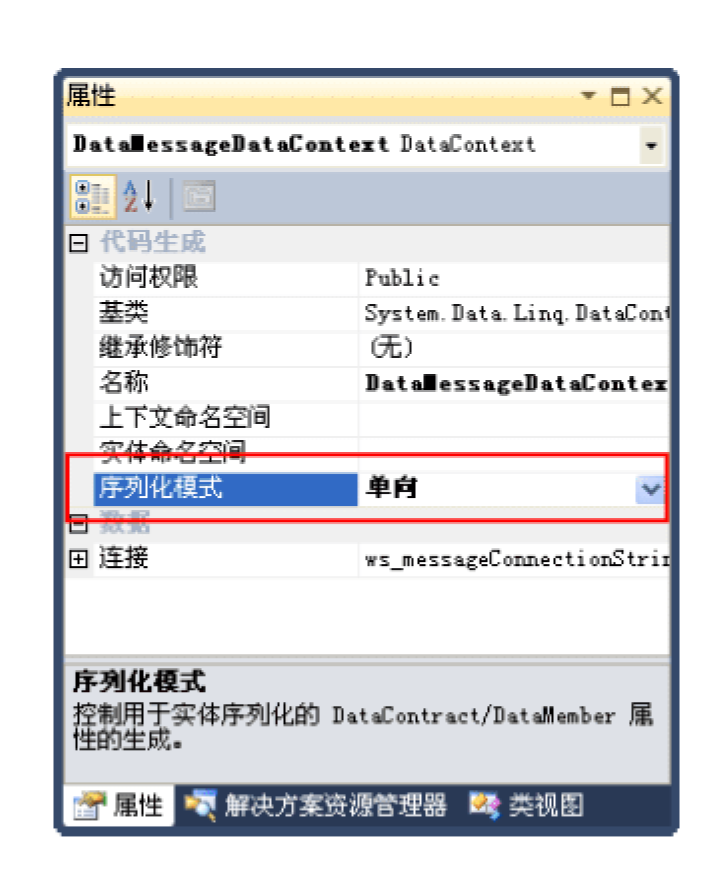


图 14-9　设置序列化模式

然后，分别设置每一个关联关系的父属性的访问权限为 Internal，如图 14-10 所示。

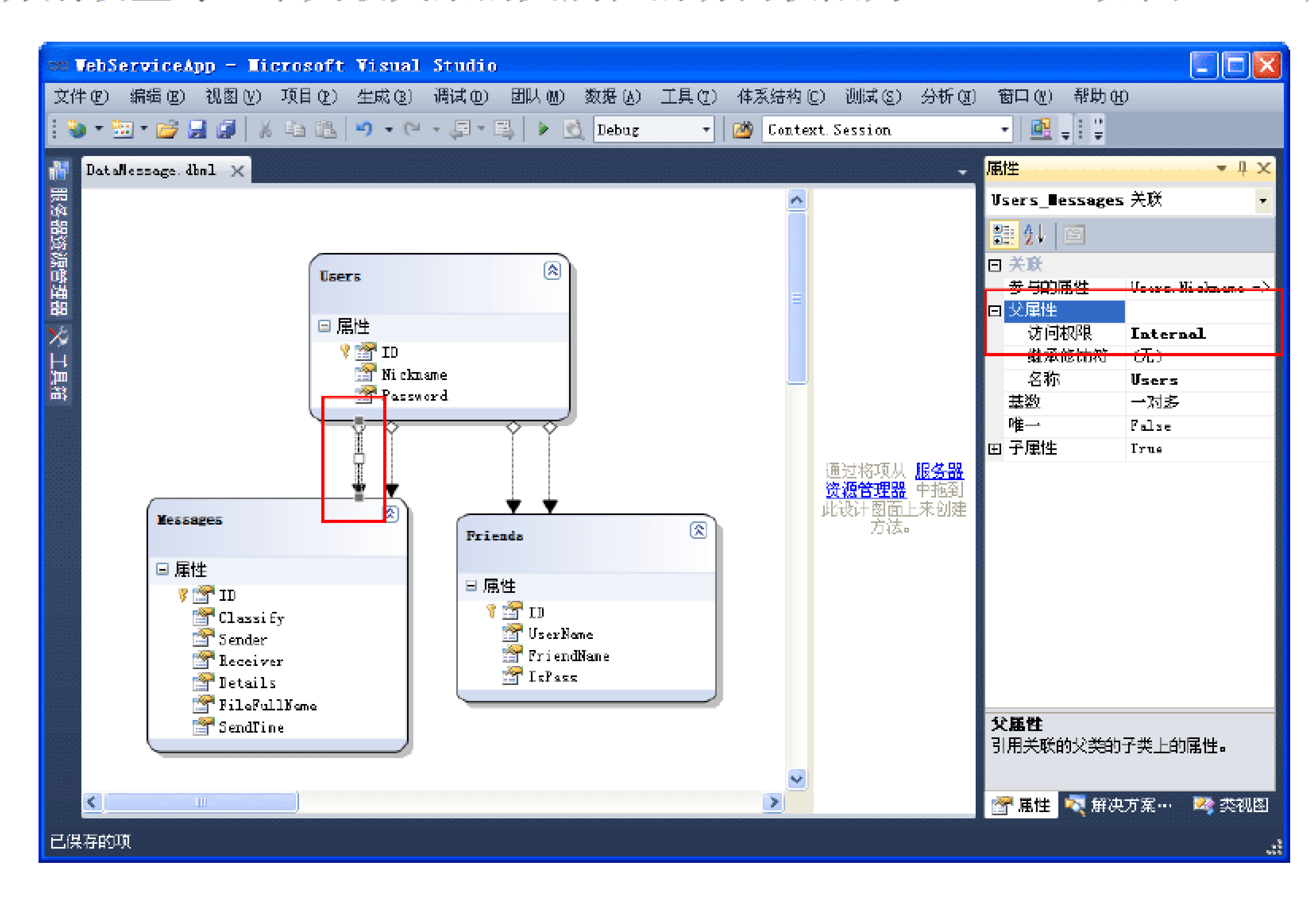


图 14-10　设置关联关系的父属性的访问权限

14.2.3　验证用户是否存在

首先需要在 Web.BLL 项目中创建一个封装业务逻辑的类，命名为 MessageManager。然后在 MessageManager 类文件中添加对 Web.DAL 项目的引用。

下面我们在 MessageManager 类中添加一个验证用户是否存在的方法，名为 Exist，实现代码如下：

```
//验证用户是否存在
public bool Exist(string nickname)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();

    //统计 Users 表中符合条件的对象的数目
    int num = dm.Users.Count(u => u.Nickname == nickname);

    //如果查询得到的数目大于 0，说明该用户存在
    bool exist = num > 0;

    return exist;
}
```

然后，需要在 Web 项目 WebServiceApp 中创建一个 Web 服务，用于向 Internet 中发布。该 Web 服务命名为 WSMessage。

同样需要在 Web 服务 WSMessage 中添加用于验证用户是否存在的 Web 方法，具体实现代码如下所示：

```
//验证用户是否存在
[WebMethod]
public bool HaveUser(string nickname)
{
    MessageManager mm = new MessageManager();
    bool exist = mm.Exist(nickname);
    return exist;
}
```

14.2.4　用户注册功能

首先需要在 Web.BLL 项目中的 MessageManager 类中添加执行注册功能的代码，具体要求代码如下：

```
//执行注册功能
public bool Register(Users user)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();

    try
    {
        dm.Users.InsertOnSubmit(user);
        dm.SubmitChanges();
    }
    catch
    {
```

```
        return false;
    }
    return true;
}
```

然后在 Web 服务 WSMessage 中添加发布注册功能的 Web 方法，具体实现代码如下：

```
//注册
[WebMethod]
public bool Register(string nickname, string password)
{
    MessageManager mm = new MessageManager();
    Users user = new Users()
    {
        Nickname = nickname,
        Password = password
    };

    return mm.Register(user);
}
```

14.2.5　用户登录功能

首先需要在 MessageManager 中添加执行登录验证功能的代码，用于根据用户名和密码返回一个用户对象。如果返回为空，即登录不成功。具体实现代码如下：

```
//执行登录功能
public Users Login(string nickname, string password)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();
    try
    {
        //查询得到指定用户
        Users user = dm.Users.Single(u => u.Nickname == nickname);
        return user;
    }
    catch
    {
        return null;
    }
}
```

同样在 Web 服务 WSMessage 中添加验证用户登录的 Web 方法，具体代码如下：

```
//验证登录
[WebMethod(EnableSession = true)]
public bool Login(string nickname, string password)
{
```

```
    MessageManager mm = new MessageManager();
    Users user = mm.Login(nickname, password);

    //状态保持
    Session["CurrentUser"] = user;

    return user != null;
}
```

14.2.6　添加好友功能

首先，在 MessageManager 类中加入用于添加好友功能的业务逻辑代码。这里我们需要过滤用户的多次请求，如果某个用户向指定的用户发送多次好友请求，服务器只记录一次。执行该操作的时候需要同时向消息表和好友信息表中写入数据库。具体实现代码如下：

```
//添加好友
public void AddFriend(Messages msg)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();
    //查询用户是否已经是好友或好友请求已经发送成功
    var fs = (from f in dm.Friends
              where f.UserName == msg.Sender && f.FriendName == msg.Receiver
              select f).ToList();
    //如果查询到相关信息，就不再执行该操作
    if (fs != null && fs.Count > 0)
    {
        return;
    }
    Friends friend = new Friends()
    {
        UserName = msg.Sender,
        FriendName = msg.Receiver,
        IsPass = false
    };
    dm.Messages.InsertOnSubmit(msg);
    dm.Friends.InsertOnSubmit(friend);
    dm.SubmitChanges();
}
```

同样需要在 Web 服务 WSMessage 中添加相应的处理代码，具体代码如下：

```
//添加好友
[WebMethod(EnableSession = true)]
public void AddFriend(string nickname)
{
    Users user = Session["CurrentUser"] as Users;
    if (user == null)
```

```
    {
        throw new Exception("用户尚未登录！");
    }
    Messages msg = new Messages();
    msg.Classify = 3;
    msg.Receiver = nickname;
    msg.SendTime = DateTime.Now;
    msg.Sender = user.Nickname;
    MessageManager mm = new MessageManager();
    mm.AddFriend(msg);
}
```

14.2.7 处理好友请求

用户接受到好友请求以后，需要对好友请求进行处理。我们可以接受请求或者拒绝请求，这两个操作的执行方式不同：在同意请求的时候需要修改好友信息的状态；在拒绝请求的时候需要删除请求信息。所以需要在 MessageManager 中分别添加接受和拒绝操作的代码，具体代码如下：

```
//同意好友请求
public void AgreeFriend(Messages msg)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();
    Friends friend = dm.Friends.Single(f =>
           f.IsPass == false &&
           f.UserName == msg.Receiver &&
           f.FriendName == msg.Sender);
    friend.IsPass = true;
    dm.Messages.InsertOnSubmit(msg);
    dm.SubmitChanges();
}
//拒绝好友请求
public void RejectFriend(Messages msg)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();
    Friends friend = dm.Friends.Single(f =>
           f.IsPass == false &&
           f.UserName == msg.Receiver &&
           f.FriendName == msg.Sender);
    dm.Friends.DeleteOnSubmit(friend);
    dm.Messages.InsertOnSubmit(msg);
    dm.SubmitChanges();
}
```

然后，还需要在 Web 服务 WSMessage 中添加处理好友请求的 Web 方法，具体代码如下：

```
//处理好友请求
[WebMethod(EnableSession = true)]
public void HandleFriendRequest(bool accept,string friend)
{
    Users user = Session["CurrentUser"] as Users;
    if (user == null)
    {
        throw new Exception("用户尚未登录！");
    }
    Messages msg = new Messages();
    msg.Classify = accept ? 4 : 5;
    msg.Receiver = friend;
    msg.SendTime = DateTime.Now;
    msg.Sender = user.Nickname;
    MessageManager mm = new MessageManager();
    if (accept)
    {
        mm.AgreeFriend(msg);
    }
    else
    {
        mm.RejectFriend(msg);
    }
}
```

14.2.8　发送消息功能

首先，需要在 MessageManager 类中添加发送消息功能的方法，用于向数据库中保存一条消息，具体代码如下：

```
//发送信息
public void SendMessage(Messages msg)
{
    if (msg.Classify == 3)
    {
        AddFriend(msg);
    }
    else if (msg.Classify == 4)
    {
        AgreeFriend(msg);
    }
    else if (msg.Classify == 5)
    {
        RejectFriend(msg);
    }
    //创建一个数据环境对象
```

```
    DataMessageDataContext dm = new DataMessageDataContext();
    dm.Messages.InsertOnSubmit(msg);
    dm.SubmitChanges();
}
```

然后需要在 Web 服务 WSMessage 中添加相应的发送消息功能的 Web 方法，具体代码如下：

```
//发送消息
[WebMethod(EnableSession = true)]
public void SendMessage(Messages msg)
{
    Users user = Session["CurrentUser"] as Users;
    if (user == null)
    {
        throw new Exception("用户尚未登录！");
    }
    msg.Classify = 1;
    msg.SendTime = DateTime.Now;
    msg.Sender = user.Nickname;
    MessageManager mm = new MessageManager();
    mm.SendMessage(msg);
}
```

14.2.9 发送文件功能

因为在服务器端，文件信息对象和普通消息对象的执行操作相同，所以这里可以使用 MessageManager 类中发送消息的方法来执行发送文件操作。

但是在 Web 服务 WSMessage 中还要创建一个 Web 方法，具体代码如下：

```
//发送文件
[WebMethod(EnableSession = true)]
public void SendFile(Messages msg)
{
    Users user = Session["CurrentUser"] as Users;
    if (user == null)
    {
        throw new Exception("用户尚未登录！");
    }
    msg.Classify = 2;
    msg.SendTime = DateTime.Now;
    msg.Sender = user.Nickname;

    MessageManager mm = new MessageManager();
    mm.SendMessage(msg);
}
```

14.2.10　监听信息功能

首先我们需要在 MessageManager 类中添加查询指定用户消息的方法，具体代码如下：

```
//获取信息列表
public List<Messages> GetMessages(string nickname)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();
    //查询得到符合条件的结果集
    var ms = from m in dm.Messages
              where m.Receiver == nickname
              select m;
    //将结果集转换成 List
    List<Messages> messages = ms.ToList();
    //为了不让下面的删除操作影响查询结果集，我们在这里把该结果集复制一份
    messages = CopyMessageList(messages);
    //删除数据库中已被用户获取过的结果集
    dm.Messages.DeleteAllOnSubmit(ms);
    dm.SubmitChanges();
    //返回结果
    return messages;
}
```

因为需要同时执行查询和删除操作，删除操作可能会影响查询操作的结果，我们在执行删除操作以前需要将查询结果复制到一个集合中，以备使用。所以这里还需要一个复制查询结果集的私有方法，具体代码如下：

```
//Copy 信息列表
private List<Messages> CopyMessageList(List<Messages> msgs)
{
    List<Messages> messages = new List<Messages>();
    foreach (var m in msgs)
    {
        Messages msg = new Messages();
        msg.ID = m.ID;
        msg.Classify = m.Classify;
        msg.Details = m.Details;
        msg.FileFullName = m.FileFullName;
        msg.Receiver = m.Receiver;
        msg.Sender = m.Sender;
        msg.SendTime = m.SendTime;
        messages.Add(msg);
    }
    return messages;
}
```

然后，还要在 Web 服务 WSMessage 中添加一个执行监听功能的 Web 方法，具体代码如下：

```
//监听信息
[WebMethod(EnableSession = true)]
public List<Messages> Monitor()
{
    Users user = Session["CurrentUser"] as Users;
    if (user == null)
    {
        throw new Exception("用户尚未登录！");
    }
    MessageManager mm = new MessageManager();
    var ms = mm.GetMessages(user.Nickname);
    return ms;
}
```

14.2.11 获取好友列表功能

首先，需要在 MessageManager 类中添加一个获取指定用户好友列表的方法，具体代码如下：

```
//获取指定用户的好友列表
public List<Friends> GetFriendList(string nickname)
{
    //创建一个数据环境对象
    DataMessageDataContext dm = new DataMessageDataContext();
    var fs = from f in dm.Friends
              where f.UserName == nickname && f.IsPass == true
              select f;
    return fs.ToList();
}
```

接下来还要在 Web 服务 WSMessage 中添加一个查询用户好友列表的 Web 方法，具体代码如下：

```
//获取好友列表
[WebMethod(EnableSession = true)]
public List<Friends> GetFriendList()
{
    Users user = Session["CurrentUser"] as Users;
    if (user == null)
    {
        throw new Exception("用户尚未登录！");
    }

    MessageManager mm = new MessageManager();
    return mm.GetFriendList(user.Nickname);
}
```

14.3 客户端设计

在前面我们也都看到了，该项目的客户端使用的是 Windows 窗体应用程序，并且在客户端一共有 4 个应用程序窗体，分别是注册窗体 FormRegister、登录窗体 FormLogin、好友查找窗体 FormFindFriend 和聊天主窗体 FormMessage。

客户端需要在服务器端保存用户的状态，所以需要在客户端添加对 Web 服务项目的 Web 引用。引用的命名空间名称为 WebServiceMessage。

14.3.1 注册窗体功能设计

前面也看过注册窗体的设计界面，现在为其添加实现程序。

窗体只实现“注册用户”的功能。具体实现方式是在用户单击注册按钮时，获取表单上用户输入的信息，进行一些简单的验证，然后调用 Web 服务方法执行注册功能，注册成功时关闭当前窗体。具体实现代码如下：

```
private void btnSubmit_Click(object sender, EventArgs e)
{
    string nickname = this.txtNickname.Text.Trim();
    string password = this.txtPassword.Text.Trim();
    string rePassword = this.txtRePassword.Text.Trim();
    //用户名不能为空
    if (nickname == string.Empty)
    {
        MessageBox.Show(""用户名"不能为空！");
        return;
    }
    //密码不能为空
    if (password == string.Empty)
    {
        MessageBox.Show(""密码"不能为空！");
        return;
    }
    //两次密码必须一致
    if (password != rePassword)
    {
        MessageBox.Show("两次密码不一致！");
        return;
    }
    //禁用提交按钮
    this.btnSubmit.Enabled = false;
    //创建一个代理类对象
    WebServiceMessage.WSMessage wsm = new WebServiceMessage.WSMessage();
```

```
    //验证用户是否存在
    bool isExist = wsm.HaveUser(nickname);
    if (isExist)
    {
        MessageBox.Show("该用户名已存在，请换个用户名再试！");
        this.btnSubmit.Enabled = true;
        return;
    }
    //获取执行结果
    bool over = wsm.Register(nickname, password);
    //如果执行成功，提示并关闭注册对话框
    //如果执行失败，提示并让用户重试
    if (over)
    {
        MessageBox.Show("注册成功！");
        this.Close();
    }
    else
    {
        MessageBox.Show("注册失败，请重试。");
        this.btnSubmit.Enabled = true;
    }
}
```

14.3.2 登录窗体功能设计

登录窗体是应用程序运行时出现的第一个窗体。但是因为其不是主要功能，所以不能把它当做应用程序启动的默认窗体。

这里将登录窗体在应用程序默认窗体的构造方法中打开，这样虽然它不是主窗体，但是却可以作为第一个窗体打开。

但是如果在登录窗体中，用户不执行登录操作而直接关闭该窗体，应用程序也理应正常关闭。而如果用户登录成功，系统则关闭登录窗体并同时打开聊天主窗体。所以，应用程序要添加对关闭操作时的设计。代码如下：

```
public partial class FormLogin : Form
{
    private FormMessage master = null;
    private bool CloseKey = true;
    public FormMessage Master
    {
        get { return master; }
        set { master = value; }
    }
    private void FormLogin_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (CloseKey)
```

```
        {
            Application.Exit();
        }
    }

    //其他代码省略
}
```

这里添加了一个主窗体的类属性，以便登录窗体在登录成功以后能够对主窗体进行一些简单的设置。而且在登录窗体中还添加了一个名为 CloseKey 的属性，是为了区别在登录窗体关闭时，是直接关闭还是登录成功关闭，以便执行一些相应的操作。

当然，在登录窗体中还有一个【登录】按钮和一个【注册】按钮。【注册】按钮只是为了打开注册窗口来让用户执行注册操作；登录操作是执行用户的验证和登录功能。

其中【注册】按钮的单击事件处理程序如下：

```
private void btnRegister_Click(object sender, EventArgs e)
{
    //创建一个注册窗口
    FormRegister fr = new FormRegister();
    //以对话框的形式打开该窗口
    fr.ShowDialog();
}
```

【登录】按钮的单击事件处理程序如下：

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string nickname = this.txtUsername.Text.Trim();
    string password = this.txtPassword.Text.Trim();
    //用户名不能为空
    if (nickname == string.Empty)
    {
        MessageBox.Show(""用户名"不能为空！");
        return;
    }
    //密码不能为空
    if (password == string.Empty)
    {
        MessageBox.Show(""密码"不能为空！");
        return;
    }
    WebServiceMessage.WSMessage wsm = new WebServiceMessage.WSMessage();
    //创建一个 Cookie 容器，用于保存用户会话状态
    CookieContainer cookies = new CookieContainer();
    //向主窗体传递 Web 服务代理对象，以重复使用
    this.Master.WsMessage = wsm;
    wsm.CookieContainer = cookies;
    bool over = wsm.Login(nickname, password);
    if (over)
```

```
    {
        this.Master.CurrentUsername = nickname;
        this.CloseKey = false;
        //关闭登录框
        this.Close();
    }
    else
    {
        MessageBox.Show("登录失败，请重试！");
        this.btnLogin.Enabled = true;
        this.btnRegister.Enabled = true;
    }
}
```

14.3.3 添加好友窗体功能设计

添加好友窗体功能非常简单，只需要用户输入一个用户名，然后单击【发送请求】按钮，即可向服务器发送相应的好友请求信息。

在该窗体中使用了一个代理类对象，这里我们为添加好友窗体类添加一个类属性，用于得到登录窗体传递的代理类对象。

在用户单击【发送请求】按钮时，需要验证用户是否存在，如果系统中不存在该用户，提示用户并取消操作。而且还需要验证，如果用户已经是我们的好友，那么将不能添加这个用户，并提示该用户已经存在于好友列表。最后请求发送成功以后，关闭该窗体。

下面来看一下【发送请求】按钮的单击事件处理程序代码：

```
private void btnRequest Click(object sender, EventArgs e)
{
    string username = this.txtUsername.Text.Trim();
    //验证输入的用户名不能为空
    if (username == null || username == string.Empty)
    {
        MessageBox.Show("请输入要请求的用户名称。");
        return;
    }

    //判断用户是否存在
    bool isExist = WsMessage.HaveUser(username);
    if (!isExist)
    {
        MessageBox.Show("用户不存在！");
        return;
    }
    WebServiceMessage.Friends[] friends = WsMessage.GetFriendList();
    foreach (var f in friends)
    {
        if (username == f.FriendName)
```

```
        {
            MessageBox.Show("该用户已经是你的好友！");
            return;
        }
    }
    //添加好友
    WsMessage.AddFriend(username);

    this.Close();
}
```

14.3.4　聊天窗体功能设计

在聊天主窗体中主要执行五个操作：刷新好友列表、添加好友、发送文件、发送消息以及监听信息等。

该窗体类需要使用一个已登录过的 Web 服务代理对象和一个登录过的用户名，所以需要在该类中添加两个类属性，代码如下：

```
public partial class FormMessage : Form
{
    WebServiceMessage.WSMessage wsMessage = null;
    private string currentUsername = null;
    #region Public Attribute

    public WebServiceMessage.WSMessage WsMessage
    {
        get { return wsMessage; }
        set { wsMessage = value; }
    }
    public string CurrentUsername
    {
        get { return currentUsername; }
        set { currentUsername = value; }
    }
    #endregion
    //其他代码省略
}
```

聊天窗体是应用程序的主窗体，所以需要在应用程序入口点 Main 方法中修改由该窗体启动应用程序。客户端 Program 类中 Main 方法的代码如下：

```
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new FormMessage());
}
```

系统启动的时候会自动调用该窗体的构造方法，执行初始化操作。这里我们在构造方法中添加相应的初始化代码，如下所示：

```
public FormMessage()
{
    InitializeComponent();
    if (WsMessage == null)
    {
        FormLogin login = new FormLogin();
        login.Master = this;
        login.ShowDialog();
        if (CurrentUsername == null || WsMessage == null)
        {
            return;
        }
        this.Text += " (当前用户: " + this.CurrentUsername + ")";
        //初始化好友列表
        this.RefreshFriendList();

        WsMessage.Timeout = -1;
        //开始监听
        this.timerMonitor.Start();
    }
}
```



这里最后一行代码 this.timerMonitor.Start()是使用窗体中的一个 Timer 控件来启动监听程序。对于该控件的功能，稍后我们再来讲解。初始化好友列表的方法 this.RefreshFriendList()的代码也稍后再来讲解。

应用程序主窗体初始化完毕，就可以编写代码为其添加相应的功能处理方法了。

1．刷新好友列表

刷新好友列表功能是在单击窗体中的【刷新】按钮时执行的一个方法。另外窗体初始化、添加好友成功以后也需要刷新好友列表，所以我们把它保存成一个独立的方法。

【刷新】按钮的单击事件和刷新好友列表方法的代码如下所示：

```
private void btnRefresh_Click(object sender, EventArgs e)
{
    //刷新好友列表
    this.RefreshFriendList();
}
//刷新好友列表
private void RefreshFriendList()
{
    //清空好友列表
    this.lstUsers.Items.Clear();
    //请求获得好友列表
    WebServiceMessage.Friends[] friends = WsMessage.GetFriendList();
```

```
    //遍历添加用户列表
    foreach (var f in friends)
    {
        this.lstUsers.Items.Add(f.FriendName);
    }
}
```

2. 添加好友

这里的添加好友操作只需要打开“添加好友”对话框(窗体)即可，具体代码如下：

```
private void btnSearch_Click(object sender, EventArgs e)
{
    FormFindFriend fff = new FormFindFriend();
    fff.WsMessage = this.WsMessage;
    fff.ShowDialog();
}
```

3. 发送消息

发送消息功能非常简单。我们只需要把文本框中的内容封装成一个 Messages 对象，调用 Web 方法执行发送操作，并将消息添加到消息列表中，清空消息框即可。具体实现代码如下：

```
private void btnSend_Click(object sender, EventArgs e)
{
    string content = this.txtMessage.Text.Trim();
    object target = this.lstUsers.SelectedItem;
    if (content == string.Empty)
    {
        return;
    }
    if (target == null)
    {
        MessageBox.Show("请选择好友！");
        return;
    }

    WebServiceMessage.Messages msg = new WebServiceMessage.Messages();
    msg.Classify = 1;
    msg.Details = content;
    msg.Receiver = target.ToString();
    WsMessage.SendMessage(msg);
    //将发送的消息加入列表
    string message = "我对 " + msg.Receiver + " 说：\r\n  " + msg.Details;
    this.AddMessageToList(message);
    //清空文本框
    this.txtMessage.Text = string.Empty;
}
```

这里用到了一个 AddMessageToList()方法，用于将一个文本消息添加到窗体中的消息列表中，实现代码如下：

```
//将消息加入列表
private void AddMessageToList(string msg)
{
    this.txtMessages.Text += msg;
    this.txtMessages.Text += "\r\n\r\n";
}
```

4. 发送文件

发送文件操作需要让用户选择一个文件，并将用户选定的文件转换成一个字节数组，然后将该字节数组转换成一个字符串对象，使用该字符串对象作为 Messages 对象的消息内容封装，调用 Web 方法进行提交。具体代码如下：

```
private void btnSendFile_Click(object sender, EventArgs e)
{
    object target = this.lstUsers.SelectedItem;
    if (target == null)
    {
        MessageBox.Show("请选择好友！");
        return;
    }
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.ShowDialog();
    string filename = ofd.FileName;
    //如果用户没有选择文件，不执行操作
    if (filename == null || filename.Trim() == string.Empty)
    {
        return;
    }
    //限制发送文件的大小
    FileInfo file = new FileInfo(filename);
    long fileLength = file.Length;
    //如果发送的文件大于 300KB，拒绝发送(文件过大会降低应用程序性能)
    if ((fileLength / 1024) > 10)
    {
        MessageBox.Show("系统暂时不支持发送大于 10KB 的文件，请重新选择文件。");
        return;
    }
    //确认发送文件
    DialogResult result = MessageBox.Show(
        "是否发送文件：\r\n" + filename + "\r\n 大小：" + (fileLength / 1024) + "K",
        "确认发送", MessageBoxButtons.YesNo);
    if (result == DialogResult.No)
    {
        return;
    }
```

```
    //发送文件
    this.SendFile(filename, target.ToString());
}
//发送文件
private void SendFile(string filename, string target)
{
    //创建一个文件流对象，并初始化
    FileStream fs = new FileStream(filename, FileMode.Open);
    //创建一个二进制数组
    byte[] bs = new byte[fs.Length];
    //从文件流中读取内容
    fs.Read(bs, 0, bs.Length);
    //关闭流
    fs.Close();
    //初始化 Messages 对象
    WebServiceMessage.Messages msg = new WebServiceMessage.Messages();
    msg.Classify = 2;
    msg.Details = ConvertStringAndBytes.ConvertBytesToString(bs);
    msg.Receiver = target;
    msg.FileFullName = filename.Substring(filename.LastIndexOf("\\") + 1);
    //执行发送操作
    WsMessage.SendFile(msg);
}
```



这里为了防止发送文件太大浪费系统性能，限制用户发送的文件大小不得超过 10KB。

当然这里用到了一个执行字符串和二进制数组转换功能的静态方法类 ConvertStringAndBytes，该类代码如下：

```
public class ConvertStringAndBytes
{
    //将字节数组转换为十六进制字符串
    public static string ConvertBytesToString(byte[] bs)
    {
        string str = string.Empty;
        if (bs != null)
        {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < bs.Length; i++)
            {
                sb.Append(bs[i].ToString("X2"));
            }
            str = sb.ToString();
        }
        return str;
    }
    //将十六进制符号字符串转换为字节数组
    public static byte[] ConvertStringToBytes(string str)
```

```
    {
        byte[] bs = new byte[str.Length / 2];
        for (int i = 0; i < bs.Length; i++)
        {
            bs[i] = Convert.ToByte(str.Substring(i * 2, 2), 16);
        }
        return bs;
    }
}
```

5．监听信息

因为服务器端只接收客户端的请求，不能主动向客户端发送最新消息内容，所以为了使客户端及时获取最新的消息，需要不停地对服务器端的消息状态进行监听。所以我们在聊天窗体中添加了一个监听控件 timerMonitor，使该控件每隔 2 秒钟向服务器发出一次请求，查询用户的最新状态信息。

在聊天窗体的构造方法中，我们已经启动了监听控件的监听操作，下面来看看监听控件的事件处理程序，具体代码如下：

```
private void timerMonitor_Tick(object sender, EventArgs e)
{
    //获取消息列表
    WebServiceMessage.Messages[] msgs = WsMessage.Monitor();
    //遍历处理消息列表
    foreach (var m in msgs)
    {
        //根据消息类型分类处理消息
        switch (m.Classify)
        {
            case 1:
                this.GetMessage(m);
                break;
            case 2:
                this.GetFile(m);
                break;
            case 3:
                this.GetFriendRequest(m);
                break;
            case 4:
            case 5:
                this.GetFriendResponse(m);
                break;
        }
    }
}
```

在此事件处理程序中遍历服务器响应中的消息列表，分别根据消息类型调用不同的方法执行不同的操作。此处还需要有 4 个扩展方法，分别用于处理接收到文本信息、文件、好友请求、好友请求响应等类型消息的消息对象。下面来分别了解一下这几个扩展方法。

1)　GetMessage()方法

该方法用于把接收到的文本消息添加到窗体中的消息列表中，具体实现代码如下：

```
//获取文本消息
private void GetMessage(WebServiceMessage.Messages msg)
{
    string m = msg.Sender + " 说：\r\n  " + msg.Details;
    this.AddMessageToList(m);
}
```

2)　GetFile()方法

该方法用于接收好友传过来的文件消息，并将消息中的字符串内容转换成相应的二进制文件，保存到用户磁盘中。实现代码如下：

```
//获取文件
private void GetFile(WebServiceMessage.Messages msg)
{
    DialogResult dr = MessageBox.Show(
        "好友 " + msg.Sender + "给你发来一个文件(" + msg.FileFullName +
        ", Size:" + msg.Details.Length + "Byte)。\r\n 是否接收该文件？",
        "接收文件", MessageBoxButtons.YesNo);
    //如果用户不接收该文件，放弃本次文件传递
    if (dr != DialogResult.Yes)
    {
        return;
    }
    //设置文件名称
    string filename = this.GetNewFilename(msg.FileFullName);
    byte[] fileContent = ConvertStringAndBytes. ConvertStringToBytes
          (msg.Details);
    //创建一个文件流对象，并初始化
    FileStream fs = new FileStream(filename, FileMode.OpenOrCreate);
    //向文件流中写入内容
    fs.Write(fileContent, 0, fileContent.Length);
    //关闭流
    fs.Close();
}
```

这里用到了一个用于获取用户设置文件名的方法 GetNewFilename()，该方法代码如下：

```
//设置接收的文件的文件名
private string GetNewFilename(string filename)
{
    //设置一个本地文件名
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.CheckFileExists = false;         //设置对话框不检查文件是否存在
    ofd.FileName = filename;             //设置【文件名】文本框的初始值
    ofd.ShowDialog();
    string fn = ofd.FileName;
```

```
    if (fn.Substring(1, 2) != @":\")
    {
        MessageBox.Show("选择的文件名格式不正确，请重新选择！");
        fn = this.GetNewFilename(filename);
    }
    return fn;
}
```

3)　GetFriendRequest()

该方法用于向用户询问好友请求的处理方式，并向服务器反馈用户的处理结果。具体实现代码如下：

```
//获取好友请求
private void GetFriendRequest(WebServiceMessage.Messages msg)
{
    DialogResult result = MessageBox.Show(
        "用户 "+msg.Sender+" 请求加你为好友，是否同意？",
        "好友请求", MessageBoxButtons.YesNo);
    WsMessage.HandleFriendRequest(result == DialogResult.Yes, msg.Sender);
}
```

4)　GetFriendResponse()

处理好友请求响应结果的方法非常简单，只需要接收相应的信息，然后提示用户处理结果即可。该操作可能会影响用户好友的列表，所以在方法末尾更新一下窗体中的用户列表信息。具体实现代码如下：

```
//获取好友请求响应
private void GetFriendResponse(WebServiceMessage.Messages msg)
{
    if (msg.Classify == 4)
    {
        MessageBox.Show("用户 " + msg.Sender + " 同意了你的好友请求。");
    }
    else
    {
        MessageBox.Show("用户 " + msg.Sender + " 拒绝了你的好友请求。");
    }
    //刷新好友列表
    this.RefreshFriendList();
}
```

14.4　运 行 结 果

首先运行 Web 服务项目，然后运行两次客户端应用程序，打开两个应用程序窗口，实现信息互发。



在客户端窗体项目 Message 下的 Bin 目录下的 Debug 目录中双击 Message.exe 应用程序，就可以运行窗体程序。

单击【登录】窗口中的【注册】按钮，打开【用户注册】对话框。使用【用户注册】对话框注册两个用户，用户名分别为 Tom 和“李丽”。注册成功后系统会提示注册成功，如图 14-11 所示。

分别用这两个账户登录系统，登录成功以后界面如图 14-12 所示。



图 14-11　用户注册成功

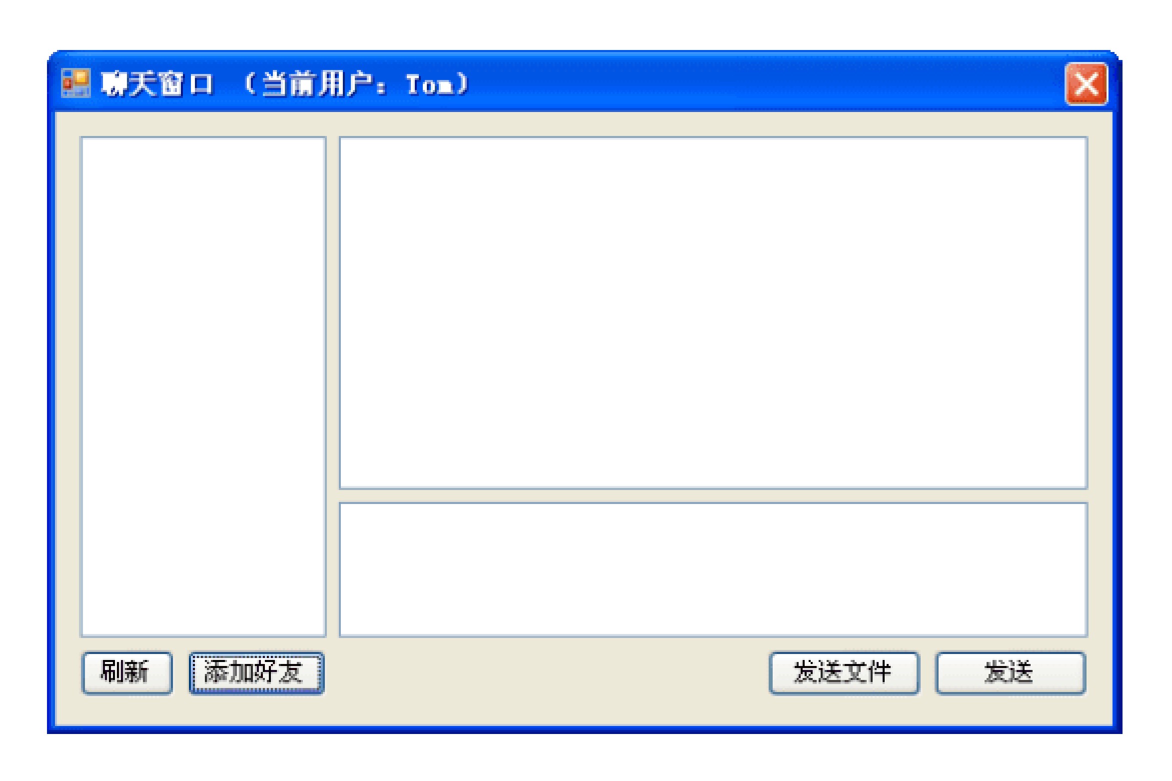


图 14-12　聊天程序主界面

在用户 Tom 的主界面中单击【添加好友】按钮，打开【添加好友】对话框，如图 14-13 所示。

在【用户名】文本框中输入要请求的好友的名字“李丽”，然后单击【发送请求】按钮，系统会自动发送好友请求并关闭【添加好友】对话框。

与此同时，用户“李丽”的聊天窗口将会弹出【好友请求】确认框，如图 14-14 所示。



图 14-13　【添加好友】对话框

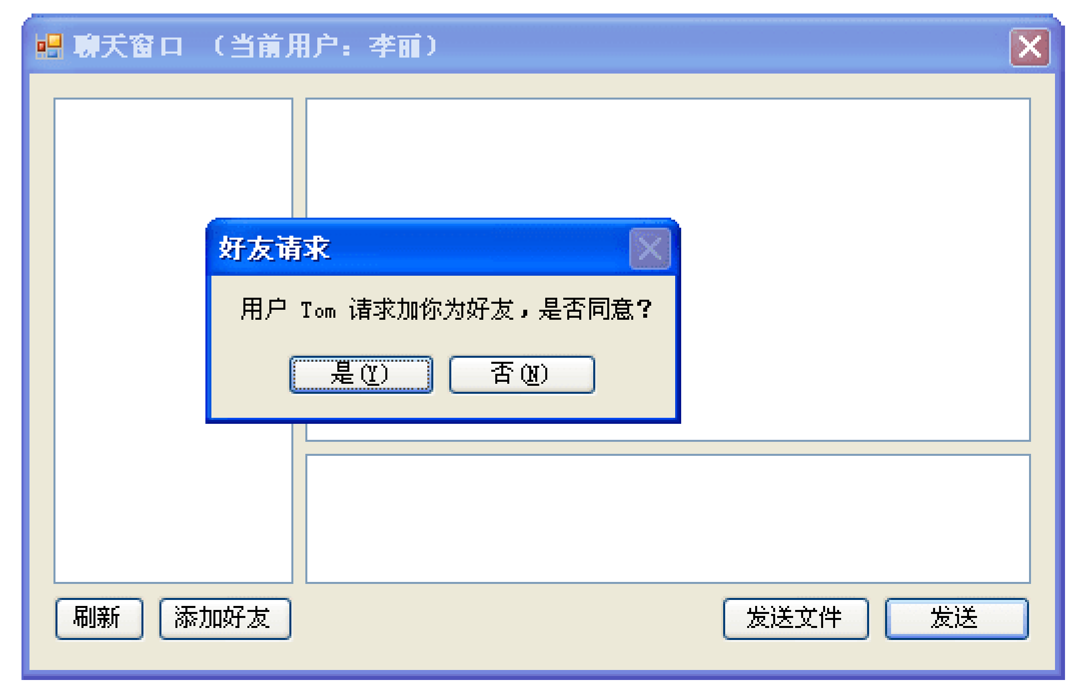


图 14-14　【好友请求】确认框

单击【是】按钮，接受该次好友请求。

同时，系统将会提示 Tom 用户已同意你的好友添加请求，如图 14-15 所示。

单击【确定】按钮，关闭提示对话框，则聊天主界面的好友列表中将自动刷新出最新的好友列表信息，如图 14-16 所示。



图 14-15　提示对话框

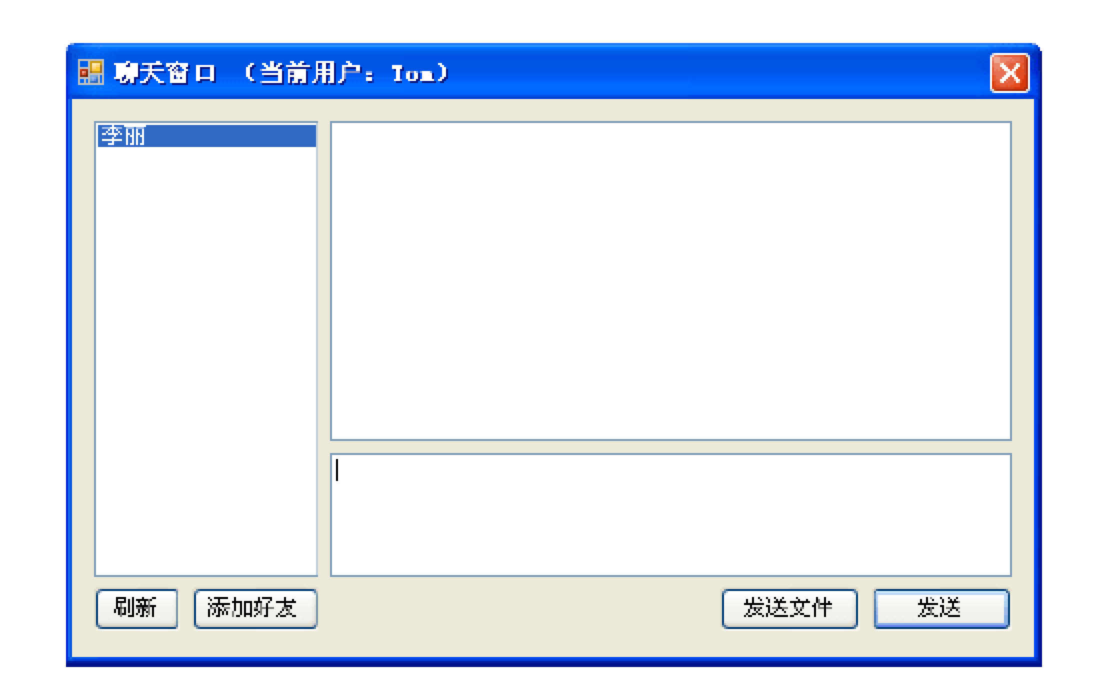


图 14-16　刷新好友列表

以同样的方式，将用户 Tom 添加到用户“李丽”的好友列表。

在用户“李丽”的好友列表中选中好友 Tom，并向其发送文本消息。比如输入“Hello Tom!”，然后单击【发送】按钮，系统会成功发送这条信息，结果如图 14-17 和图 14-18 所示。

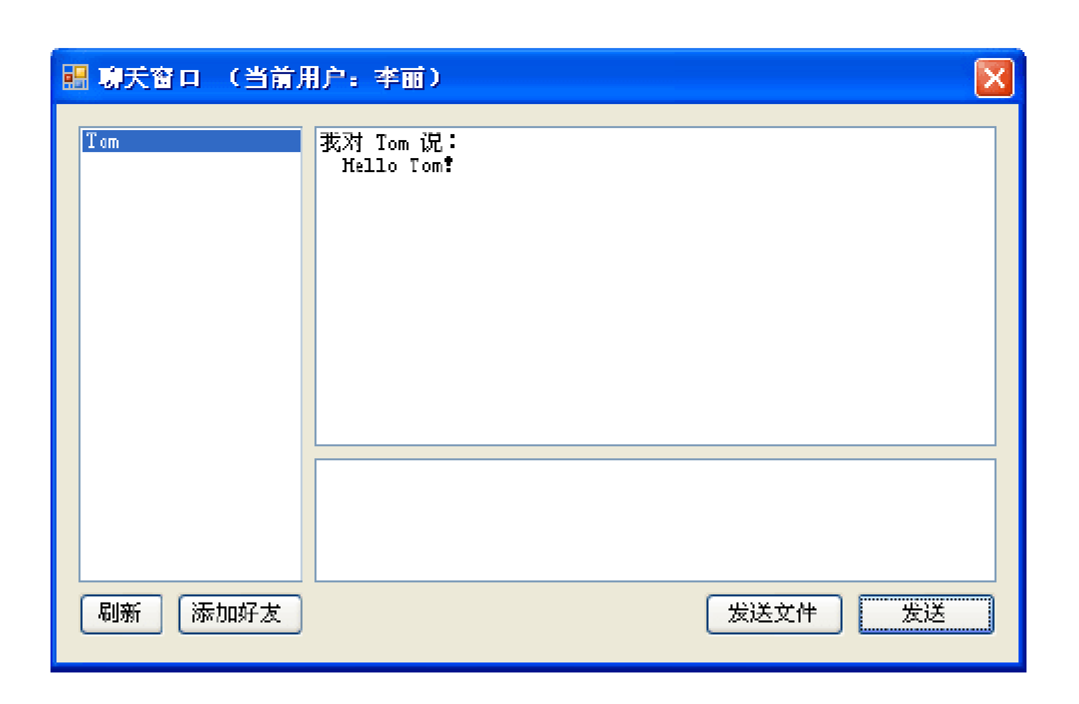


图 14-17　用户“李丽”的界面

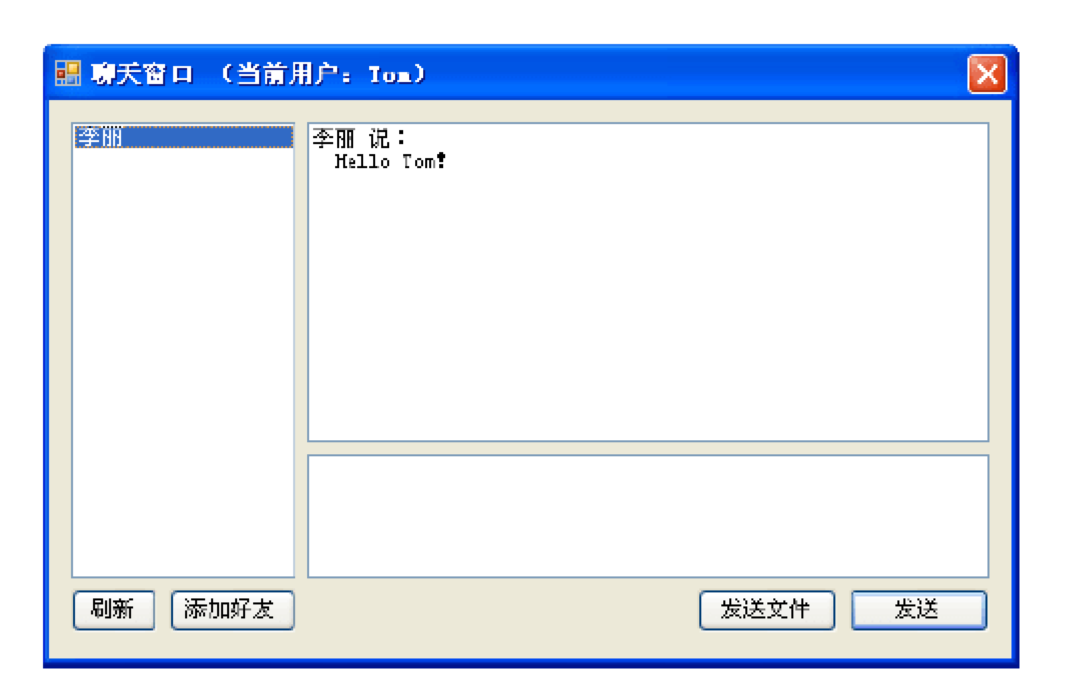


图 14-18　用户 Tom 的界面

当然，Tom 也可以向“李丽”回复消息，如图 14-19 和图 14-20 所示。

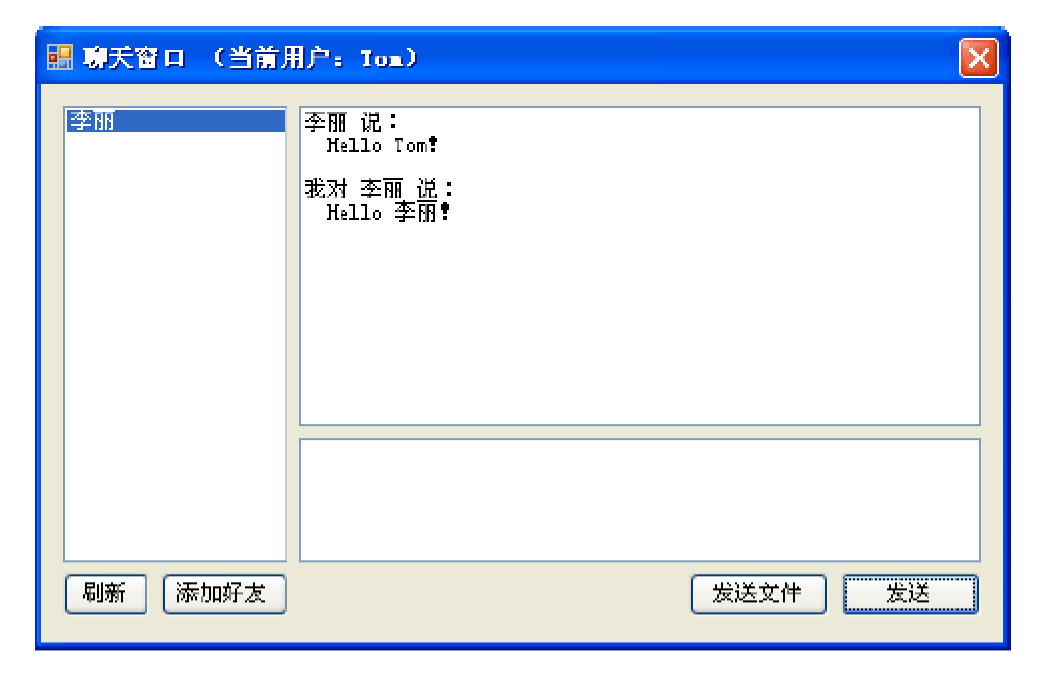


图 14-19　用户 Tom 的界面

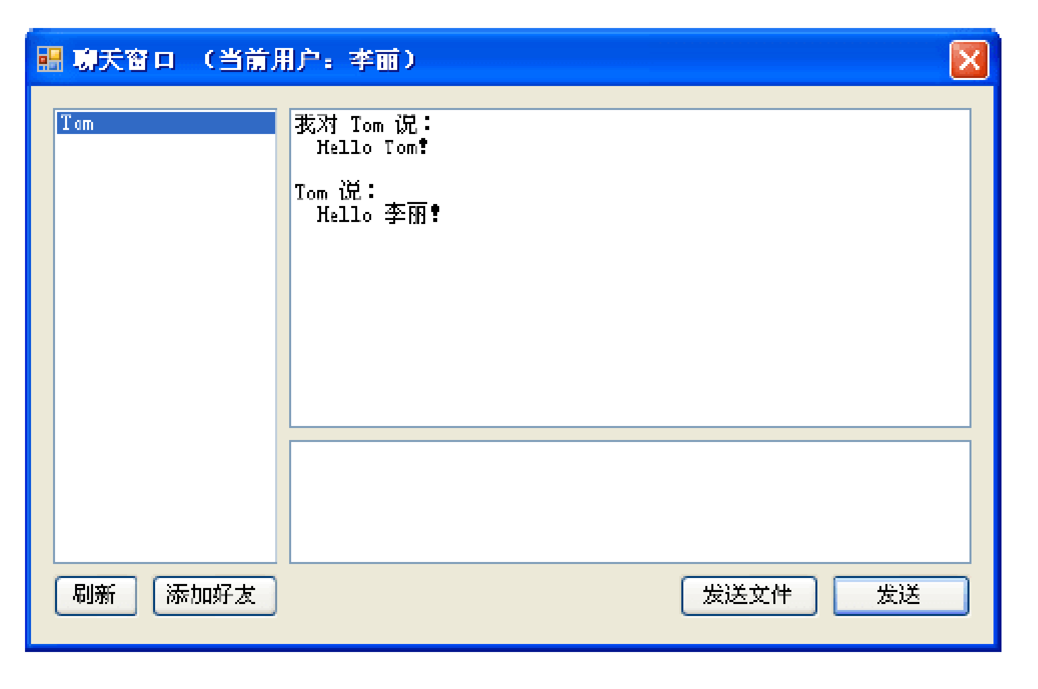


图 14-20　用户“李丽”的界面

下面来测试发送文件的功能。在用户“李丽”的主窗口中单击【发送文件】按钮，在弹出的【打开】对话框中选择一个文件，然后单击【打开】按钮，系统会询问是否向发送该文件，如图 14-21 所示。

在对话框中单击【是】按钮，系统会发送该文件。发送成功以后，用户 Tom 的窗口中就会显示“接收文件”的提示信息，如图 14-22 所示。





图 14-21　【确认发送】对话框　　　　图 14-22　【接收文件】提示框

单击【是】按钮，确认接收该文件，并在弹出的对话框中选择一个路径(比如这里将文件保存到 D 盘的新建文件夹中)。

接收以后就可以在 Windows 资源管理器中使用预览方式查看该图片了，如图 14-23 所示。



图 14-23　预览图片

14.5　总　　结

本章介绍的网络聊天工具虽然是一个比较简单的项目，但也比较系统地对 Web 服务的一些常用技能进行了一次练习。

通过这次练习，我们对 Web 服务有了更加深入的了解，并熟悉了使用 Web 服务请求数据、使用 Web 服务发送文件、使用 Web 服务保存状态等方面的技术，以及其各种技术的综合运用。

希望大家能通过这个练习学到一些知识。



第 15 章 留 言 簿

内容摘要：

留言簿是互联网上最常见的人与人之间交流的工具，或者说是客户与商家之间交流的平台。留言簿提供了一个信息交流的空间，客户可以通过留言簿向商家提出问题，对某个产品发表自己的看法。

本章我们将使用 SQL Server 2008 结合 C#语言来制作一个基于 Web 服务的 ASP.NET 应用程序——留言簿。

学习目标：

- 掌握留言簿的基本功能
- 熟悉留言簿的基本结构
- 掌握创建数据库表的方法
- 掌握 Web 网站的创建
- 掌握 Web 服务与网站的交互

15.1 项 目 概 述

本章我们开发的留言簿系统的功能有：普通用户可以浏览全部留言，不需登录就可以发表留言；管理员登录后可以对留言进行管理，包括回复留言及删除留言等。本留言簿系统将基于ASP.NET 和 Web 服务来开发。

15.1.1 功能介绍

经常上网的朋友对于留言簿可能并不陌生。留言簿是一个交流平台，浏览者可以通过留言簿和管理者进行交流。浏览者留下自己的相关资料和意见后，这些内容会被保存到数据库中，并且能够读取到页面上。留言簿可以很简单也可以很复杂，这完全由开发者的能力以及具体情况而定。

本系统是一个基本的简单型留言簿，用户可以发表自己的看法和意见，也可以查看别人的留言，管理员可以登录后对留言进行回复和删除。

根据以上分析，在系统中留言簿的基本功能主要包括如下 3 个部分。

- 用户留言：用户在网站中浏览时，对某一问题发表自己的观点和看法，并填写个人的信息，方便管理员与其他人员相互交流。
- 查看留言：网站管理者或其他人员可以对已经存在的留言进行查看、浏览，并以个人观点进行评论。
- 管理留言：对用户发表的留言进行回复和删除操作，此操作只有在管理员登录后才可以进行。

15.1.2 结构介绍

根据功能分析，可以将留言簿划分为首页、留言发表、留言管理和登录 4 个系统模块。其中，留言管理和登录这两个系统模块由管理员后台操作时使用。留言簿的 4 个系统模块及其功能描述如表 15-1 所示。

表 15-1 留言簿模块

模 块	功 能
首页	显示所有留言，管理员还可以对其中的留言进行回复
留言发表	用户发表留言
留言管理	管理员对留言进行删除或查看
登录	管理员登录

留言簿的各个模块均由一个或多个页面来实现，如表 15-2 所示展示了各个页面文件及其功能描述。

表 15-2　留言簿页面文件

模　块	功　能
listMessage.aspx	显示所有留言，管理员登录后可以对其中的留言进行删除操作
addMessage.aspx	用户发表留言
regMessage.aspx	管理员对留言进行回复操作
Login.aspx	管理员登录

15.1.3　自定义类

为了提高代码的利用率，减少代码冗余，在本项目中将关于访问数据库的操作单独写入一个类中，命名为 DBHelper 类。该类中关于数据库连接字符串的主要实现代码如下所示：

```
//获得数据库连接
private static SqlConnection connection;
public static SqlConnection Connection
{
    get
    {
        string connectionString = "Data Source=.;Initial Catalog=message;User
          ID=sa;pwd=123";
         if (connection == null)
        {
           connection = new SqlConnection(connectionString);
           connection.Open();
        }
        else if (connection.State == System.Data.ConnectionState.Closed)
        {
           connection.Open();
        }
         else if (connection.State == System.Data.ConnectionState.Broken)
         {
            connection.Close();
            connection.Open();
         }
         return connection;
     }
}
```

ExecuteCommand()方法主要用于增加、删除、修改操作，该方法同时提供一个重载方法，代码如下：

```
/// <summary>
/// 执行 Sql 语句，并返回受影响的行数
/// </summary>
/// <param name="safeSql"></param>
```

```
/// <returns></returns>
public static int ExecuteCommand(string safeSql)
{
    SqlCommand cmd = new SqlCommand(safeSql, Connection);
    int result = cmd.ExecuteNonQuery();
    return result;
}
public static int ExecuteCommand(string sql, params SqlParameter[] values)
{
    SqlCommand cmd = new SqlCommand(sql, Connection);
    cmd.Parameters.AddRange(values);
    int result = cmd.ExecuteNonQuery();
    return result;
}
```

下列的 4 个方法主要用于查询时调用，返回不同的结果，代码如下：

```
/// <summary>
/// 执行查询，返回 SqlDataReader 集合
/// </summary>
/// <param name="safeSql"></param>
/// <returns></returns>
public static SqlDataReader GetReader(string safeSql)
{
    SqlCommand cmd = new SqlCommand(safeSql, Connection);
    SqlDataReader reader = cmd.ExecuteReader();
    return reader;
}

public static SqlDataReader GetReader(string sql, params SqlParameter[] values)
{
    SqlCommand cmd = new SqlCommand(sql, Connection);
     cmd.Parameters.AddRange(values);
     SqlDataReader reader = cmd.ExecuteReader();
     return reader;
}

/// <summary>
///执行查询，返回 DataSet 集合
/// </summary>
/// <param name="safeSql"></param>
/// <returns></returns>
public static DataSet GetDataSet(string safeSql)
{
    DataSet ds = new DataSet();
    SqlCommand cmd = new SqlCommand(safeSql, Connection);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    da.Fill(ds);
    return ds;
}
```

```
public static DataSet GetDataSet(string sql, params SqlParameter[] values)
{
    DataSet ds = new DataSet();
    SqlCommand cmd = new SqlCommand(sql, Connection);
    cmd.Parameters.AddRange(values);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    da.Fill(ds);
    return ds;
}
```

15.2　数据库设计

对留言簿进行结构分析后，就要开始设计用于存储留言信息的数据库了。本系统使用的数据库是 SQL Server 2008，数据库名 message。

由于留言簿相对简单，本数据库包括两个表，一个是留言信息表 messageInfo，一个是管理员表 adminInfo。messageInfo 表主要用于存储留言信息，包括用户留言和管理员回复信息；adminInfo 表存储的是管理员信息，包括管理员名称和密码。两个表的数据结构及说明如表 15-3 和表 15-4 所示。

表 15-3　adminInfo 表

字　段	数据类型	长　度	允 许 空	备　注
id	int	4	否	标识
adminName	varchar	10	否	管理员名称
passWord	varchar	10	否	管理员密码

表 15-4　messageInfo 表

字　段	数据类型	长　度	允 许 空	备　注
id	int	4	否	标识
name	varchar	10	是	发表人名称
msgTitle	varchar	20	是	留言主题
msgContent	varchar	1000	是	留言发表内容
phohe	varchar	15	是	发表人联系方式
mail	varchar	50	是	发表人 E-Mail
isOnly	int	4	是	表示是否开启仅管理员可见(0 否 1 是)
addtime	datetime	8	是	留言发表时间
recontent	varchar	1000	是	回复内容

15.3 服务器端设计

项目结构分析和数据库设计完成后，就可以开发项目了。我们在服务器端需要实现的功能主要有显示所有留言、管理员登录、回复留言和删除留言等。

15.3.1 项目结构

该留言簿是基于 Web 服务的 ASP.NET 项目，所以需要用一个 Web 服务项目来实现需求。由于留言簿系统涉及数据库的操作，并且需要某些业务逻辑，所以这里使用三层架构的设计模式来实现项目。

创建解决方案 message，然后在解决方案里添加 Web 服务项目 MessageWeb；接着创建一个用于封装数据实体类的类库项目，命名为 model；再创建一个用于封装数据访问操作的类库项目，命名为 DAL；最后创建用于封装业务逻辑的类库项目，命名为 BLL。

该留言簿系统使用 B/S 架构设计，所以还需要向该解决方案中添加 ASP.NET 网站项目，命名为 message。

全部创建完成后，解决方案资源管理器中的结构如图 15-1 所示。

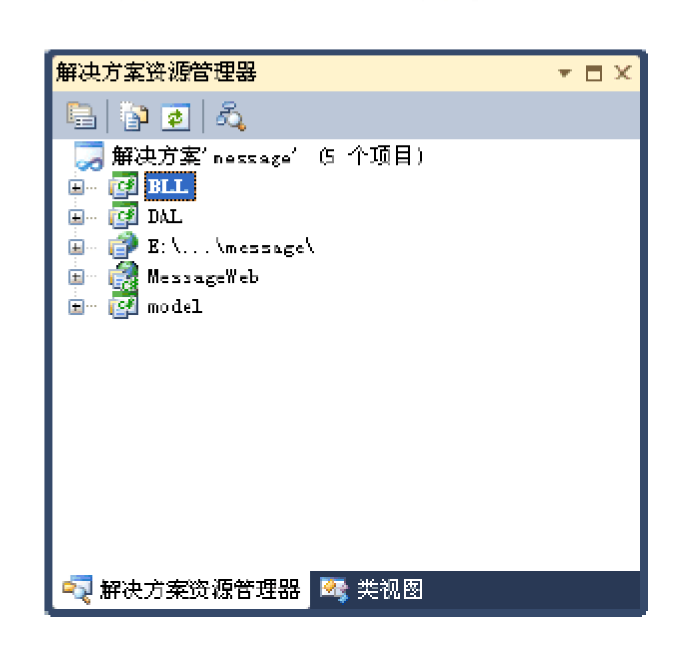


图 15-1　解决方案资源管理器结构

当然，创建好项目后不要忘了为这几个项目添加引用关系。BLL 引用 DAL 和 model；Web 服务项目 MessageWeb 引用 BLL 和 model；ASP.NET 项目 message 引用 Web 服务项目 MessageWeb。



在实际开发中，一般只有大型项目才使用三层架构的设计模式来实现。中小型项目可根据实际需求选择性地使用三层架构设计模式。

15.3.2 管理员登录功能

在名称为 BLL 的类库项目中添加 ManagerAdmin 类文件，并添加用于管理员登录验证的方法 GetAdmin，输入参数为管理员名称和密码，返回结果为 Admin 实体类。GetAdmin 方法表示根据输入的用户名和密码返回一个对象，如果返回结果为空，表示登录不成功；如果返回结果

非空，表示用户登录成功，将登录信息读取出来并保存到初始化的 Admin 实体类中，然后返回结果。具体代码如下所示：

```
/// <summary>
/// 查看当前登录管理员是否存在
/// </summary>
/// <param name="name">参数 用户名</param>
/// <param name="pwd">参数 密码</param>
/// <returns>返回结果</returns>
public admin GetAdmin(string name, string pwd)
{
    admin a = new admin();
    SqlDataReader dr = dal.GetAdmin(name, pwd);
    if (dr.Read())
    {
        a.Id = Convert.ToInt32(dr["id"]);
        a.AdminName = dr["adminname"].ToString();
        a.PassWord = dr["password"].ToString();
        dr.Close();
        return a;
    }
    else
    {
        dr.Close();
        return null;
    }
}
```

该方法中的“dal”为 ServiceAdmin 类的实例化。ServiceAdmin 类位于数据访问层 DAL，表示和数据库的交换信息。

同时，我们还要在 Web 服务 adminWebService 下面添加验证用户登录的方法，代码如下所示：

```
[WebMethod]
/// <summary>
/// 查看当前登录管理员是否存在
/// </summary>
/// <param name="name">参数 用户名</param>
/// <param name="pwd">参数 密码</param>
/// <returns>返回结果</returns>
public admin GetAdmin(string name, string pwd)
{
    admin a = admin.GetAdmin(name, pwd);
    if (a != null)
    {
        return a;
    }
    else
    {
```

```
        return null;
    }
}
```

注意，在 Web 服务中添加的方法前需要添加“[WebMethod]”，这样，该方法才能在 Web 中进行访问，否则该方法无效。

15.3.3 添加留言

在数据访问层 ServiceMessage 类中的添加留言的方法名称为 add，输入参数为 message 实体类，返回值为 int 类型，即执行 SQL 语句后返回的受影响的行数。实现代码如下所示：

```
/// <summary>
/// 添加一条留言
/// </summary>
/// <param name="message"></param>
/// <returns></returns>
public int add(message message)
{
   string sql = "INSERT INTO messageInfo (name, msgTitle, msgContent, phone,
mail, isOnly,datetime,recontent) VALUES
(@name,@title,@content,@phone,@mail,@isonly,@datetime,@recontent)";
SqlParameter[] para = new SqlParameter[]{
       new SqlParameter("@name",message.Name),
       new SqlParameter("@title",message.MsgTitle),
       new SqlParameter("@content",message.MsgContent),
       new SqlParameter("@phone",message.Phone),
   new SqlParameter("@mail",message.Mail),
       new SqlParameter("@isonly",message.isOnly),
   new SqlParameter("@datetime",message.Datetime),
   new SqlParameter("@recontent",message.Recontent),
   };
   try
   {
    return DBHelper.ExecuteCommand(sql, para);
   }
   catch (Exception e)
   { return 0; }
}
```

在业务逻辑层 BLL 中，ManagerMessage 类添加对添加留言的操作代码，方法名为 add，如下所示：

```
/// <summary>
/// 添加一条留言
/// </summary>
/// <param name="message"></param>
/// <returns></returns>
public int add(message message)
```

```
{
    return dal.add(message);
}
```

在 add 方法中的 dal 为数据访问层 ServiceMessage 类的实例化，然后调用 ServiceMessage 类的添加留言方法，因为这两个方法定义的返回值相同，所以可以直接返回结果。

同样，还需要在 Web 服务中实现 add 方法，在 Web 服务 messageService 中添加如下代码：

```
[WebMethod]
/// <summary>
/// 添加一条留言
/// </summary>
/// <param name="message"></param>
/// <returns></returns>
public int add(message message)
{
    return new BLL.ManagerMessage().add(message);
}
```

15.3.4　获得留言列表

添加留言时，可以选择是否开启仅管理员可见，在数据库中表示该项功能的字段是 isOnly，功能描述为是否开启仅管理员可见(0 表示否，1 表示是)。所以，在获得留言列表时，有两种形式：当管理员登录时，将所有留言按照时间的先后顺序读取；当普通用户浏览留言时，将显示 isOnly 字段为 0 的所有留言。

在 ManagerMessage 中添加代码用来实现获得留言列表，代码如下：

```
/// <summary>
/// 获得所有留言信息
/// </summary>
/// <returns></returns>
public DataSet GetMessage()
{
    string sql = "select * from messageInfo order by datetime desc";
    return dal.GetMessage(sql);
}
/// <summary>
/// 获得没有限制管理员可见的所有留言
/// </summary>
/// <returns></returns>
public DataSet GetMessageByIsRead()
{
    string sql = "select * from messageInfo where isonly=0 order by datetime
desc";
    return dal.GetMessage(sql);
}
```

我们定义了两个方法 GetMessage 和 GetMessageByIsRead。方法 GetMessage 表示获得所有留言，方法 GetMessageByIsRead 表示如果是普通用户查看留言，那么仅显示 isOnly 标记为 0 的留言，标记为 1 的留言表示仅管理员可见。

在 ManagerMessage 类中的方法里出现的 dal 则是 ServiceMessage 类的实例化对象，ServiceMessage 位于数据访问层。ServiceMessage 类中的 GetMessage 方法的实现代码如下所示：

```
/// <summary>
/// 获得所有留言信息
/// </summary>
/// <returns></returns>
public DataSet GetMessage(string sql)
{
    DataSet ds = new DataSet();
    try
    {
        ds = DBHelper.GetDataSet(sql);
        return ds;
    }
    catch(Exception e)
    {
        return null;
    }
}
```

在 ServiceMessage 类中 GetMessage 方法接收的参数为一条 SQL 查询语句，返回一个 DataSet 集合。

同样地，我们还需要在 Web 服务中实现业务逻辑层 ManagerMessage 类中定义的两个查询方法。在 Web 服务 messageService 里添加如下代码：

```
[WebMethod]
//获得留言列表
public DataSet GetAllMessage(int isOnly)
{
    if (isOnly == 1)
    {
        return new BLL.ManagerMessage().GetMessage();
    }
    else if (isOnly == 0)
    {
        return new BLL.ManagerMessage().GetMessageByIsRead();
    }
    else
    {
        return null;
    }
}
```

在 GetAllMessage 方法中，参数 isOnly 用来表示当前的登录状态。当 isOnly 为 1 时，表示为登录状态，调用函数获得所有留言信息；当 isOnly 为 0 时，表示未登录状态，调用函数获得

所有 isOnly 标记为 0 的留言信息。数据库中的 isOnly 字段只有 0 和 1 两个值，如果参数为其他，则返回一个空值。

15.3.5　管理留言功能

管理员对留言的管理包括对留言进行删除和回复两个功能。

在数据访问层 DAL 的 ServiceMessage 类中，定义了删除方法 DelMessage 和回复方法 AddRegMsg。删除方法 DelMessage 接收参数留言 ID，返回执行 SQL 语句后受影响的行数。回复方法 AddRegMsg 接收参数 message 实体类，根据留言 ID 更新数据表 recontent 的内容，返回执行 SQL 语句后受影响的行数。

位于数据访问层 DAL 的 ServiceMessage 类的具体实现代码如下所示：

```
/// <summary>
/// 添加回复
/// </summary>
/// <returns></returns>
public int AddRegMsg(message message)
{
    string sql = "UPDATE messageInfo SET recontent =@recontent WHERE (ID = @id)";
    try
    {
        SqlParameter[] para = new SqlParameter[] {
            new SqlParameter("@recontent",message.Recontent),
            new SqlParameter("@id",message.Id)
        };
        return DBHelper.ExecuteCommand(sql, para);
    }
    catch (Exception e)
    { return 0; }
}

/// <summary>
/// 删除留言
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public int DelMessage(int id)
{
    string sql = "DELETE FROM messageInfo WHERE (ID = @id)";
    try
    {
        return DBHelper.ExecuteCommand(sql, new SqlParameter("@id", id));
    }
    catch (Exception e)
    { return 0; }
}
```

接下来是在业务逻辑层的实现。在 ManagerMessage 中添加实现代码，如下所示：

```
/// <summary>
/// 删除留言
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public int DelMessage(int id)
{
    return dal.DelMessage(id);
}
/// <summary>
/// 添加回复
/// </summary>
/// <returns></returns>
public int AddRegMsg(message message)
{
    return dal.AddRegMsg(message);
}
```

同样地，还需要在 Web 服务 messageService 中添加如下代码：

```
[WebMethod]
/// <summary>
/// 添加回复
/// </summary>
/// <returns></returns>
public bool AddRegMsg(message message)
{
    int num = new BLL.ManagerMessage().AddRegMsg(message);
    if (num != 0)
    {
        return true;
    }
    else
    {
         return false;
     }
 }
 [WebMethod]
 /// <summary>
 /// 删除留言
 /// </summary>
 /// <param name="id"></param>
 /// <returns></returns>
 public bool DelMessage(int id)
 {
     int num = new BLL.ManagerMessage().DelMessage(id);
     if (num != 0)
     {
```

```
            return true;
        }
        else
        {
            return false;
        }
    }
```

15.4　客户端设计

该留言簿系统使用的客户端是 ASP.NET 网站项目。之前的章节中，通过结构介绍了解到客户端共有 4 个页面，分别是 listMessage.aspx、addMessage.aspx、regMessage.aspx 和 Login.aspx。

15.4.1　添加留言

在添加留言页面 addMessage.aspx 中，需要添加到数据库中的数据有用户昵称、留言主题、留言内容、联系电话、电子邮箱这几项，还有一个单选按钮，表示该条留言是否开启仅管理员可见，如图 15-2 所示。

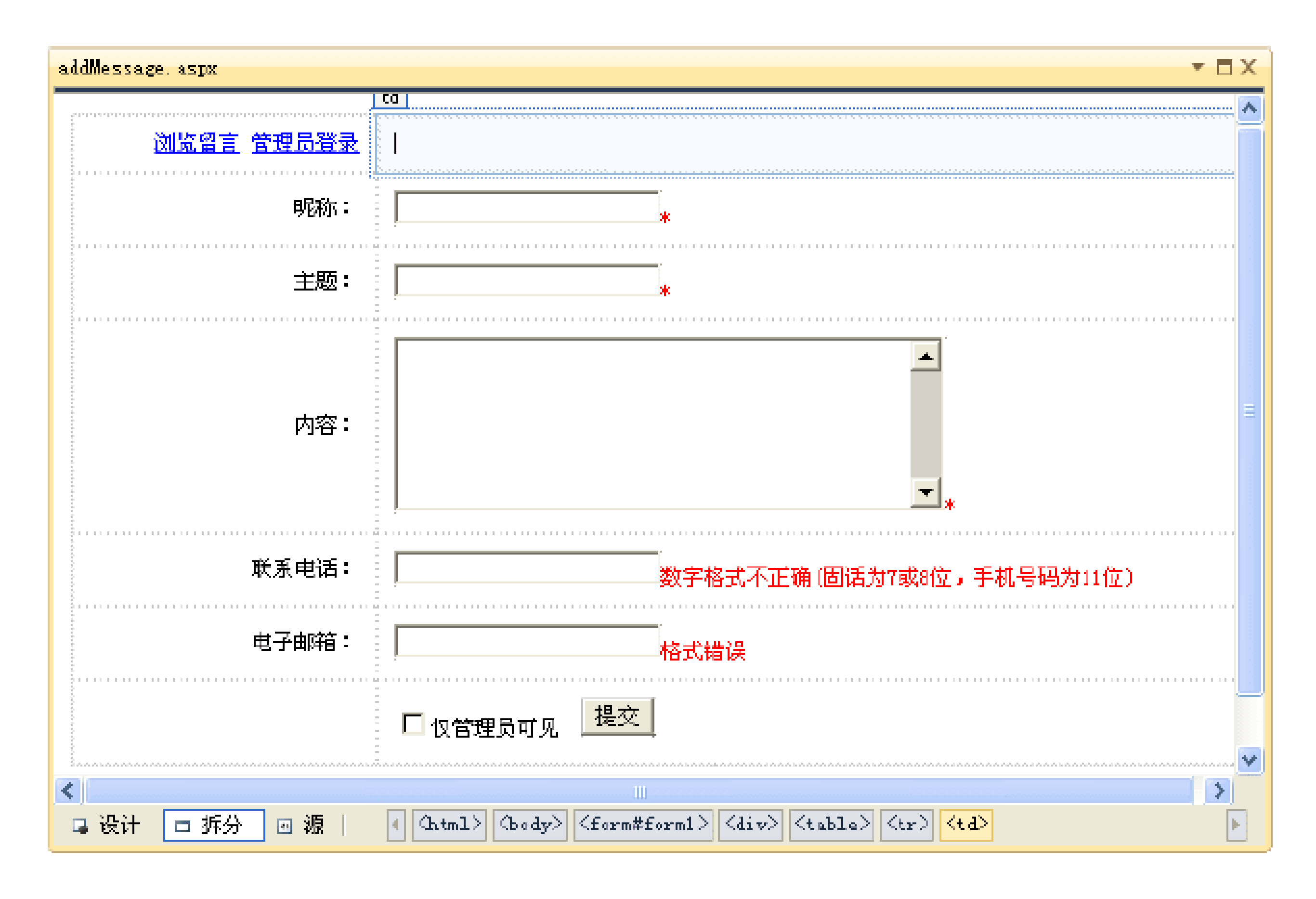


图 15-2　添加留言页面

在这里，留言内容使用的是 TextBox 控件，需要将 TextBox 控件的 TextMode 属性改为“MultiLine”。【仅管理员可见】复选框默认不启用。

为了防止有人恶意添加留言，导致系统崩溃，我们还需要添加数据验证，可以使用 ASP.NET 提供的验证控件进行验证。

昵称、主题、内容这三项输入的信息使用 RequiredFieldValidator 非空验证控件。为了防止用户留下不完整的信息，使用非空验证控件用来验证当用户提交时输入的信息是否为空。

电子邮箱使用 RegularExpressionValidator 验证控件，也称为正则表达式验证控件，它用来验证电子邮箱输入的格式是否正确。正则表达式控件的 ValidationExpression 属性可以用来输入

正则表达式，系统默认的有验证邮箱的正则表达式，当然，我们也可以自己写正则表达式。

联系方式同样使用 RegularExpressionValidator 正则表达式验证控件，用来验证输入的联系电话号码是否符合要求。

输入的内容都通过验证后，执行提交按钮的单击事件，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //提交留言
    message model = new message();
    model.Name = txtName.Text.Trim();
    model.MsgTitle = txtTitle.Text.Trim();
    model.MsgContent = txtContent.Text.Trim();
    model.Phone = txtPhone.Text.Trim();
    model.Mail = txtMail.Text.Trim();
    model.Datetime = DateTime.Now;
    model.Recontent = "暂无回复";
    //0 表示否，1 表示是
    if (CheckBox1.Checked)
    {
        model.isOnly = 1;
    }
    else
    {
        model.isOnly = 0;
    }
    WebMessage.messageService msg = new WebMessage.messageService();
    if (msg.add(model) > 0)
    {
       //回到留言列表
      Response.Redirect("listMessage.aspx");
    }
    else
    {
      Page.ClientScript.RegisterStartupScript(this.GetType(), "",
"<script>alert('添加失败！')</script>");
   }
}
```

使用的添加方法来自 Web 服务 messageService，添加成功后，页面直接跳转到 ListMessage.aspx 主页面。

15.4.2 管理员登录

Login.aspx 页面用于管理员登录，前台页面布局包括两个文本框和一个按钮，文本框分别命名为 txtAdmin 和 txtPwd，用来输入管理员名称和密码。单击【登录】按钮，根据用户名和密码查看数据库是否存在该用户，代码如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    WebAdmin.adminWebService a = new WebAdmin.adminWebService();
    admin admin = a.GetAdmin(txtAdmim.Text, txtPwd.Text);
    if (admin != null)
    {
        Session["admin"] = admin;
        Response.Redirect("listMessage.aspx");
    }
    else
    {
        Page.ClientScript.RegisterStartupScript(this.GetType(), "",
             "<script>alert('登录失败，用户名或密码错误！')</script>");
    }
}
```

在按钮的单击事件中使用 Web 服务 adminWebService 提供的方法 GetAdmin，如果存在该用户，则需要保存到 Session 中，并且转到 listMessage.aspx 页面。

15.4.3　显示和管理留言

listMessage.aspx 页面是主页面，如果在未登录状态下浏览该页面，则只显示数据库中所有的留言信息；如果管理员登录，则会显示删除和回复功能。页面前台布局如图 15-3 所示。



图 15-3　listMessage.aspx 页面布局

1．显示留言

在页面的 Load 事件里，绑定显示所有留言信息，并且判断是否有管理员登录，如果管理员登录，那么将显示管理员的名称，代码如下：

```
WebAdmin.admin admin = new WebAdmin.admin();
WebMessage.messageService webmessage = new WebMessage.messageService();
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        BindData();//绑定数据

        if (Session["admin"] != null)
        {
            lbtnamdin.Visible = false;
            lbladmin.Visible = true;
            admin = (admin)Session["admin"];
            lbladmin.Text = admin.AdminName;
        }
    }
}
```

在 Load 事件中，首先绑定数据，然后根据 Session 是否为空来判断管理员是否登录。如果不为空，则表示管理员已登录，设定管理员登录按钮 lbtnamdin 隐藏，标签 lbladmin 显示，并且从 Session 中读取出管理员信息，并将管理员名称显示到标签 lbladmin 上。

BindData 为绑定数据的自定义方法，代码如下：

```
private void BindData()
{
    DataSet ds = null;
    int isOnly = 0;
    if (Session["admin"] != null)
    {
        isOnly = 1;//开启仅管理员可见，表示获得所有留言信息
    }
    ds = webmessage.GetAllMessage(isOnly);
    PagedDataSource pds = new PagedDataSource();
    pds.DataSource = ds.Tables[0].DefaultView;
    pds.AllowPaging = true;
    AspNetPager1.RecordCount = pds.Count;
    pds.CurrentPageIndex = AspNetPager1.CurrentPageIndex - 1;
    pds.PageSize = AspNetPager1.PageSize;
    listRepeater.DataSource = pds;
    listRepeater.DataBind();
}
```

自定义局部变量 isOnly，类型为 int，默认值为 0。当管理员登录时，将 isOnly 赋值为 1，将变量 isOnly 作为参数传给方法 GetAllMessage，定义于 Web 服务的该方法，根据传入值的不同，获得不同的数据集。

在 BindData 方法中，添加了分页功能，在这里使用了 AspNetPager 分页控件。控件的前台代码如下所示：

```
<webdiyer:AspNetPager ID="AspNetPager1" runat="server"
    onpagechanged="AspNetPager1_PageChanged" AlwaysShow="True"
    FirstPageText="首页" LastPageText="尾页" NextPageText="下一页"
    PageIndexBoxType="DropDownList" PageSize="5" PrevPageText="上一页">
</webdiyer:AspNetPager>
```

使用该控件要注意，需要实现 AspNetPager 控件的 PageChanged 事件，在该事件中重新绑定 BindData 方法。代码如下所示：

```
protected void AspNetPager1_PageChanged(object sender, EventArgs e)
{
    BindData();
}
```

或者也可以实现 AspNetPager 控件的 PageChanging 事件，在该事件中则需要多添加一行代码，代码如下所示：

```
protected void AspNetPager1_PageChanging(object src,
Wuqi.Webdiyer.PageChangingEventArgs e)
{
    AspNetPager1.CurrentPageIndex = e.NewPageIndex;
    BindData();
}
```

2. 管理留言

显示留言列表信息使用的是 Repeater 数据控件，前台代码如下：

```
<asp:Repeater runat="server" ID="listRepeater"
       onitemdatabound="listRepeater_ItemDataBound"
       onitemcommand="listRepeater_ItemCommand" >
<ItemTemplate>
    <table width="650" cellpadding="8" cellspacing="1" bgcolor="#a9cfe4">
       <tr>
        <td width="150" >昵称：<%#Eval("name") %></td>
        <td>主题：<%#Eval("msgtitle") %></td>
        <td>发表时间：<%#Eval("datetime") %></td>
       </tr>
      <tr>
        <td  width="150" colspan="3">内容：  <br/>
            <%#Eval("msgcontent") %></td>
       </tr>
       <tr>
        <td width="150" colspan="3">回复内容：  <br/>
            <%#Eval("recontent") %></td>
       </tr>
       <tr>
        <td  colspan="3" align="right">
            <asp:LinkButton ID="lbtnDel" runat="server" CommandName="del"
               CommandArgument='<%#Eval("id") %>'  Visible="false">删除
               </asp:LinkButton> 
```

```
                <asp:LinkButton ID="lbtnrep" runat="server" CommandName="hui"
CommandArgument='<%#Eval("id") %>'  Visible="false">回复</asp:LinkButton>
          <br/>
             </td>
        </tr>
     </table>
     </ItemTemplate>
</asp:Repeater>
```

在 Repeater 控件里放入的删除和回复按钮，Visible 属性默认为不显示，按钮的 CommandName 属性分别设置为“del”和“hui”，用于在 Repeater 控件的 ItemCommand 事件里区分两个控件。按钮的 CommandArgument 属性绑定的是留言 id。ItemCommand 事件的代码如下所示：

```
protected void listRepeater_ItemCommand(object source,
RepeaterCommandEventArgs e)
{
    if (e.CommandName == "del")
    {
        if (bll.DelMessage(Convert.ToInt32(e.CommandArgument)) > 0)
        {
            Page.ClientScript.RegisterStartupScript(this.GetType(), "",
             "<script>alert('删除成功！')</script>");
            BindData();
        }
        else
        {
            Page.ClientScript.RegisterStartupScript(this.GetType(), "",
             "<script>alert('删除失败！')</script>");
        }
    }
    if (e.CommandName == "hui")
    {
        int id = Convert.ToInt32(e.CommandArgument);
        Response.Redirect("regMessage.aspx?id=" + id);
    }
}
```

在 ItemCommand 事件里，“e”表示 Repeater 控件本身，使用 CommandName 属性找到定义的两个值，然后进行判断，如果是“del”，那么进行删除操作，删除成功重新绑定数据，删除方法的参数从 CommandArgument 属性获取。“hui”表示回复操作，页面直接连接到 regMessage.aspx 页面，并且将参数 id 通过 Request 方式传递到该页面。

在这里，因为使用的方法都是 Web 服务提供的，所以，如果有方法返回的是实体类，接收时引用的命名空间不能是在解决方案中的类库项目 model 里的 admin 类，而是要引用 WebAdmin，才能保证程序的正确性。

因为将管理和浏览留言放在了同一页面，所以，需要判断当前管理员是否登录，然后才能显示回复和删除功能。这一判断需要在 Repeater 控件的 ItemDataBound 事件里进行，它表示在

数据绑定后激发，具体代码如下：

```
protected void listRepeater_ItemDataBound(object sender,
RepeaterItemEventArgs e)
{
    if (Session["admin"] != null)
    {
        //如果管理员登录，显示编辑项
        LinkButton lbtnDel = (LinkButton)e.Item.FindControl("lbtnDel");
        LinkButton lbtnrep = (LinkButton)e.Item.FindControl("lbtnrep");
        lbtnDel.Visible = true;
        lbtnrep.Visible = true;
    }
}
```

需要注意的是 Repeater 控件的两个事件：ItemDataBound 事件和 ItemCommand 事件。经常会有人混淆这两个事件，不知道什么时候要用哪个事件。其实很简单，ItemDataBound 事件是在数据绑定时激发，从英语的语法来讲，它是一个进行式，当数据全部绑定结束，该事件也完成了。而 ItemCommand 事件则是在数据绑定结束后，简单来讲就是一个过去式，激发控件中任一按钮时触发该事件，具体表现在对数据进行增、删、改操作时，需要在该事件中进行。

15.4.4　回复留言

在单击回复留言时，页面会跳转到回复留言页面 regMessage.aspx 进行操作。

首先，要注意回复功能只有管理员登录后才能操作，所以要对该页面进行验证。在 regMessage.aspx 页面的 Load 事件里添加如下代码：

```
WebMessage.messageService webmessage = new WebMessage.messageService();
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        if (Session["admin"] != null)
        {
            if (Request["id"] != null)
            {
                int messageId = Convert.ToInt32(Request["id"]);
                message msg = webmessage.GetMessageById(messageId);
                lblTitle.Text = msg.MsgTitle;
                lblName.Text = msg.Name;
            }
        }
        else
        {
            Response.Redirect("Login.aspx");
        }
    }
}
```

判断 Session 是否为空，不为空时表示当前管理员为登录状态，否则转到登录页面进行登录。其次，接收从 listMessage.aspx 页面传递过来的参数；最后，显示对要回复的留言的相关信息。

回复方法调用的是 Web 服务 messageService 提供的方法 AddRegMsg，在回复按钮的单击事件里代码如下所示：

```
protected void Button1 Click(object sender, EventArgs e)
{
    //回复
    if (Request["id"] != null)
    {
        message msg = new message();
        msg.Id = Convert.ToInt32(Request["id"]);
        msg.Recontent = txtcontent.Text;
        if (webmessage.AddRegMsg(msg))
        {
            //回复成功直接转回留言列表
            Response.Redirect("listMessage.aspx");
        }
        else
        {
            Page.ClientScript.RegisterStartupScript(this.GetType(), "",
             "<script>alert('回复失败！')</script>");
        }
    }
}
```

根据 AddRegMsg 方法返回的值判断回复留言是否成功，成功后直接转回 listMessage.aspx 页面。

15.5 运 行 效 果

运行 listMessage.aspx 页面，在未登录时浏览该页面，如图 15-4 所示。

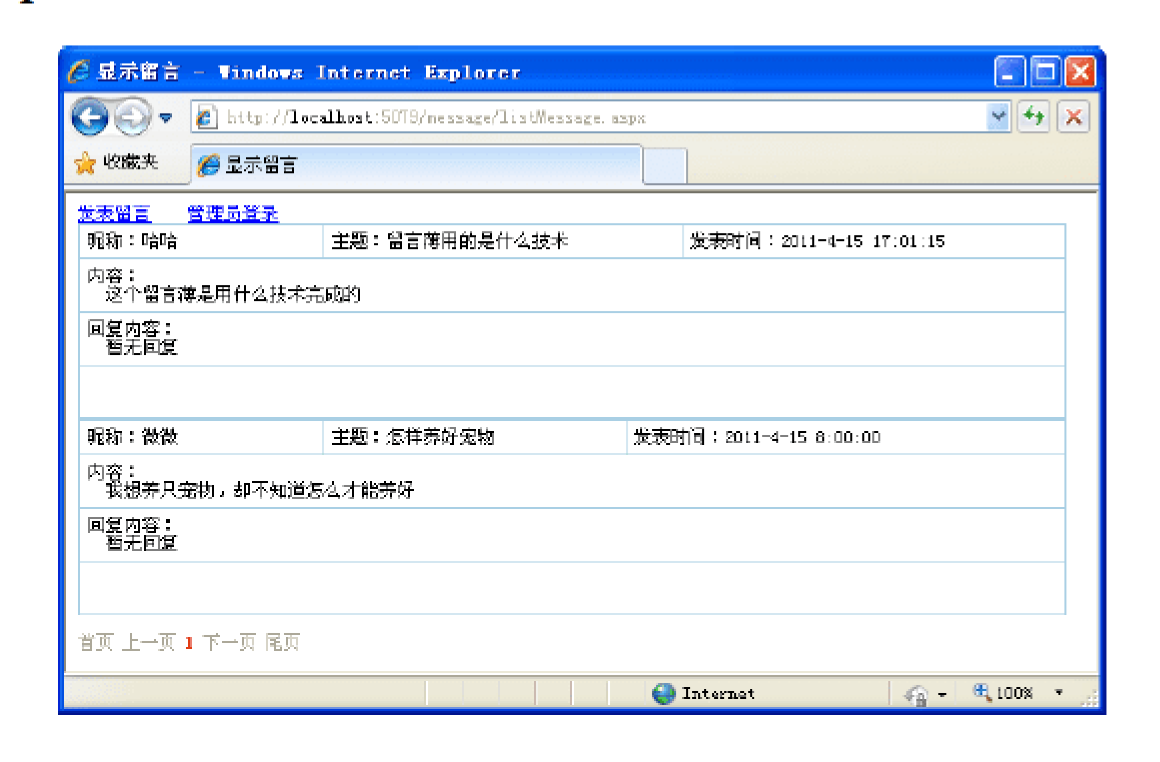


图 15-4　listMessage.aspx 初始页面

单击【发表留言】链接，跳转到 addMessage.aspx 页面，添加一条数据并选择仅管理员可见，如图 15-5 所示。

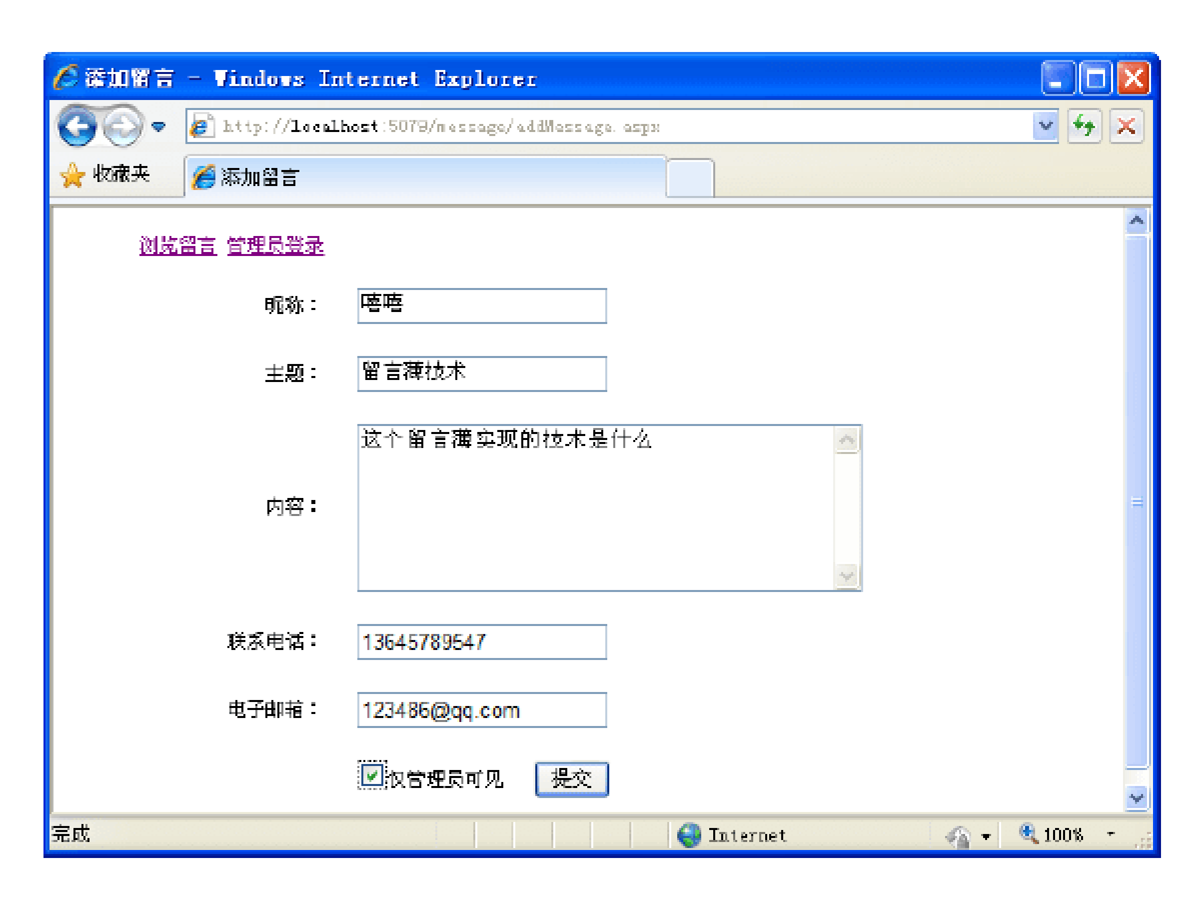


图 15-5　添加留言页面

提交后页面转回 listMessage.aspx 页面，且该条留言未显示。

单击【管理员登录】链接，使用管理员账号登录，如图 15-6 所示，用户名和密码分别是 admin 和 admin，登录后转到 listMessage.aspx 页面。这时，删除和回复功能在页面显示，如图 15-7 所示。



图 15-6　管理员登录页面

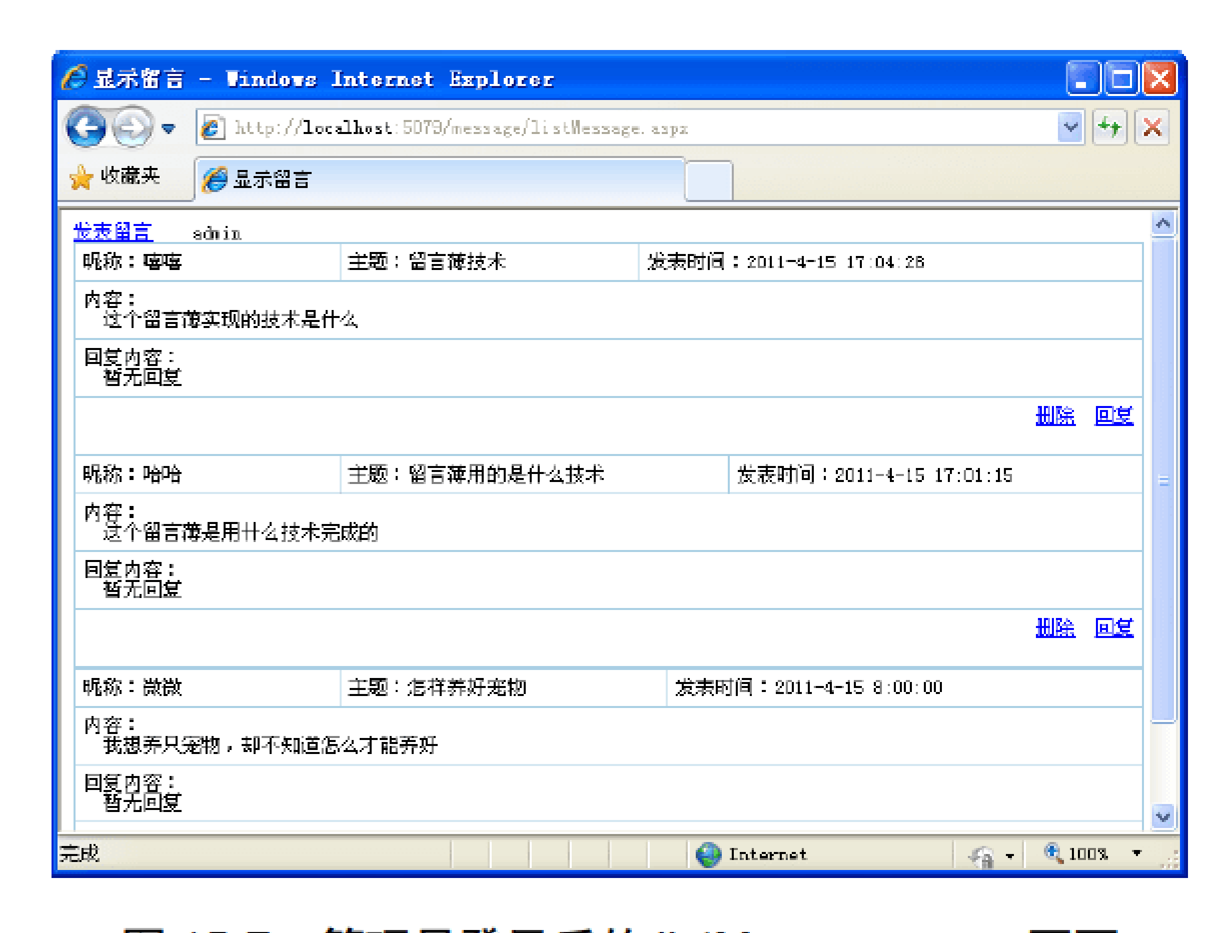


图 15-7　管理员登录后的 listMessage.aspx 页面

这时，我们开始添加的那条昵称为“嘻嘻”的留言显示在页面上，单击【回复】链接，转到 regMessage.aspx 页面，管理员进行回复，如图 15-8 所示。

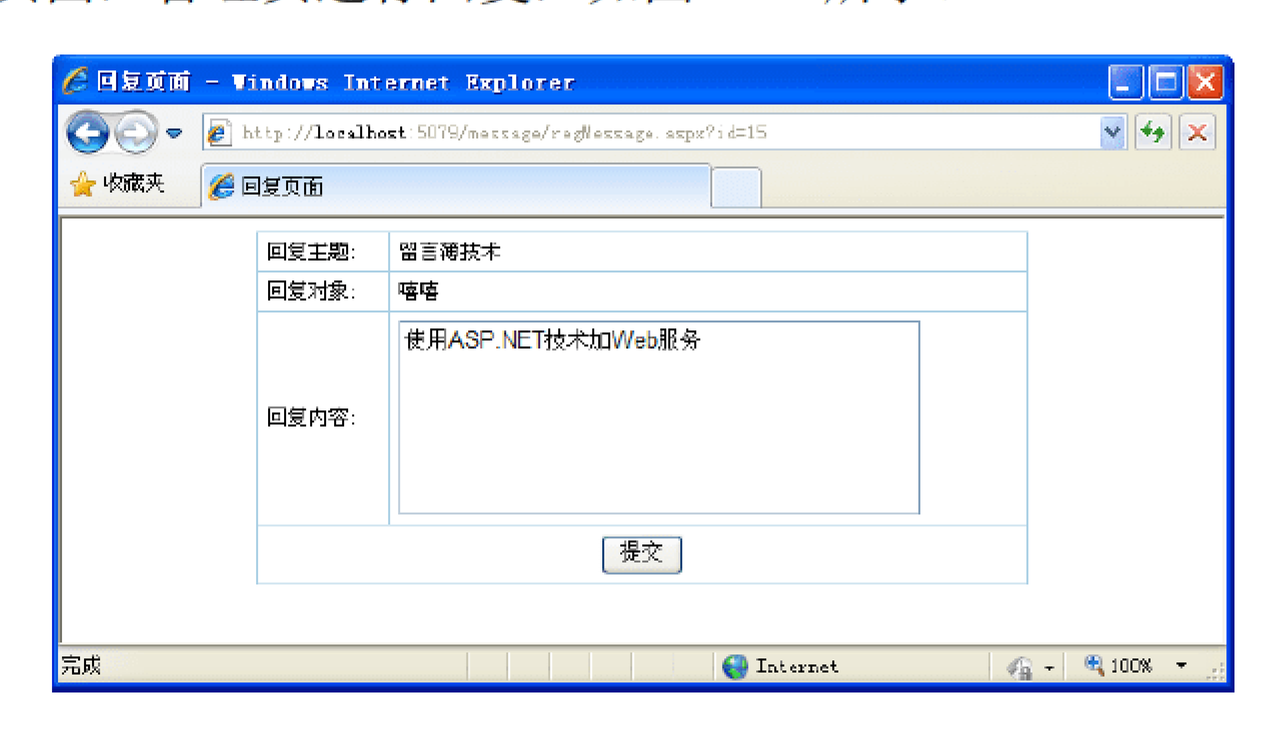


图 15-8　回复页面

管理员提交后，回到 listMessage.aspx 页面。

选择昵称为“哈哈”的留言，单击【删除】链接，进行删除操作，删除结果如图 15-9 所示。

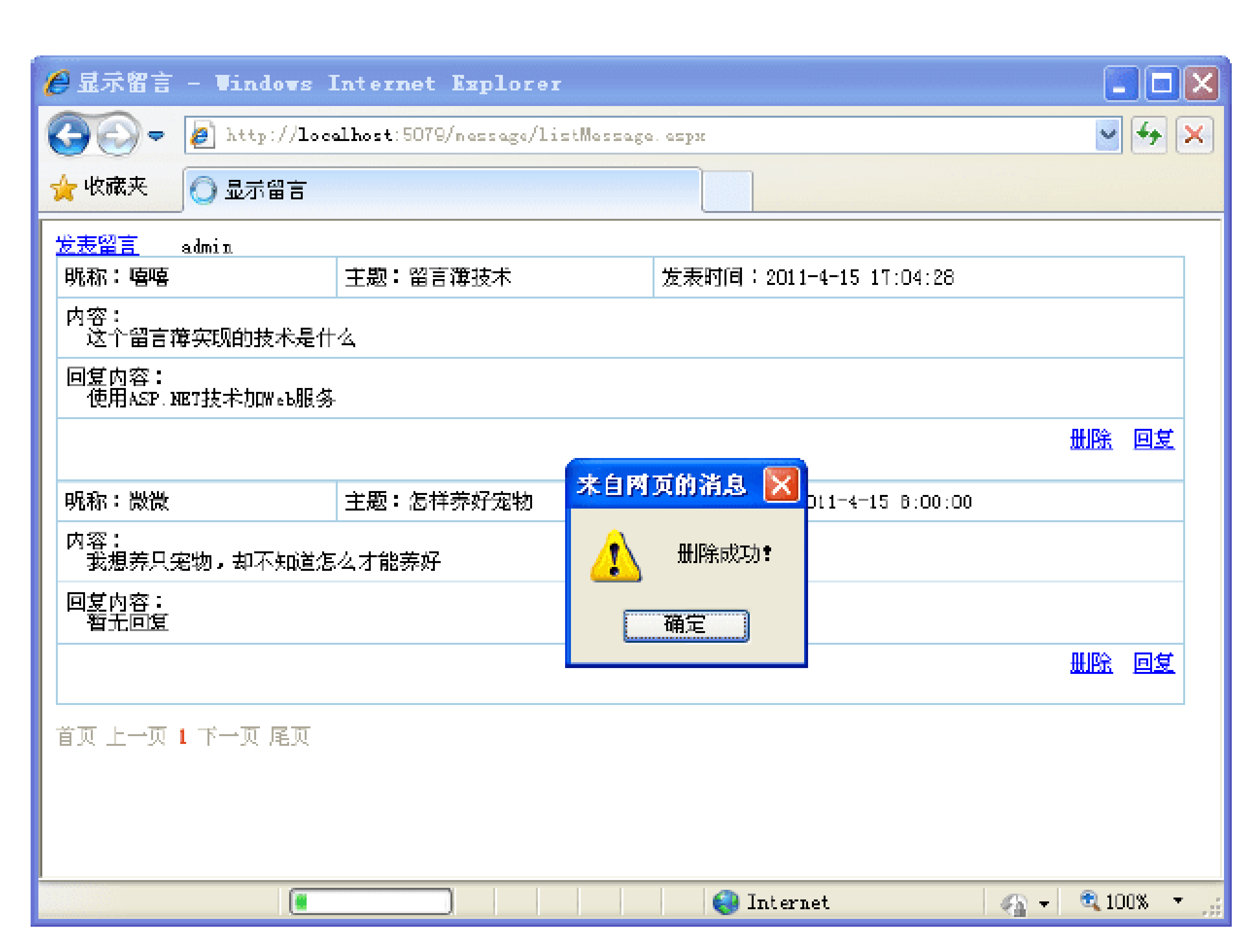


图 15-9　删除操作

15.6　总　　结

到这里，本章所介绍的留言簿系统就完成了。因为不是专业美工，所以界面可能没有网上的那些留言簿华丽，但是留言簿要求的基本功能我们还是大致完成了。

在这里，我们所实现的留言簿系统只是完成了一些简单的基本功能，有兴趣的读者可以根据具体的业务需求进行功能扩展。

通过本案例的开发，我们可以更加深刻地了解 Web 服务的实际应用。同时了解开发一个完整项目所需要的具体步骤，加深读者对软件开发的认识。

附录　各章习题参考答案

第 1 章

一、填空题

1．应用程序接口

2．SOAP

3．XML

二、选择题

1．D

2．D

3．C

第 2 章

一、填空题

1．WebMethod

2．DLL

3．wsdl.exe

4．WSDL

5．wsdl /l:cs /n:aSpace http://localhost/test.asmx

6．Web Service、System.Web.Services

二、选择题

1．A

2．C

3．A

4．C

5．A

6．A

7．B

第 3 章

一、填空题

1．standalone
2．xmlns
3．>
4．CDATA
5．!DOCTYPE
6．EMPTY
7．参数实体

二、选择题

1．B
2．A
3．C
4．D

第 4 章

一、填空题

1．W3C XML Schema
2．http://www.w3.org/2001/XMLSchema
3．32
4．<xs:element name="年龄" type="xs:integer"/>
5．Default
6．ref
7．extension、restriction
8．complexType、group

二、选择题

1．D
2．D
3．A
4．A
5．C
6．A
7．A

第 5 章

一、填空题

1．http://localhost:1025/s/test.asmx?WSDL
2．definitions
3．服务实现定义
4．type
5．getTime、getTimeResponse
6．targetNameSpace

二、选择题

1．A
2．B
3．C
4．D
5．D
6．A
7．D

第 6 章

一、填空题

1、Envelope
2、反序列化
3、SoapExtensionAttribute
4、ExtensionType
5、ProcessMessage

二、选择题

1、B
2、D
3、C

第 7 章

一、填空题

1、Session

2、sessionState

3、Application

4、Expires

二、选择题

1、B

2、C

3、C

第 8 章

一、填空题

1、生成异步操作

2、BeginInsert

3、System.IAsyncResult

二、选择题

1、C

2、B

3、D

第 9 章

一、填空题

1. 部分页缓存

2. System.Web.Caching、this.Context.Cache

3. 应用程序

4. Count、Remove()

5. onRemoveCallback

6. CacheItemPriority.High、CallBack

7. 隔离性

8．RequiresNew

二、选择题

1．A
2．D
3．D
4．D
5．C
6．B
7．A

第 10 章

一、填空题

1、集成 Windows 验证
2、消息级(端对端)安全性
3、IP
4、IIS

二、选择题

1、C
2、D
3、D
4、B

第 11 章

一、填空题

1．使用 XmlTextReader 类读取
2．DocumentSource
3．ISerializable
4．[NonSerializabled()]

二、选择题

1．C
2．A
3．B
4．A

第 12 章

一、填空题

1．图片字节流
2．AutoPostBack=“true”

二、选择题

1．A
2．D

第 13 章

一、填空题

1．.NET Framework 3.0
2．Host
3．地址、绑定
4．ServiceContract、OperationContract
5．HTTP、MSMQ
6．Open()、Abort()、Close()

二、选择题

1．A
2．D
3．B
4．A
5．C
6．D
7．A